AMSTRAD

MISE AU POINT DES PROGRAMMES BASIC

CLAUDE VIVIER/YVON JACOB



AMSTRAD

MISE AU POINT DES PROGRAMMES BASIC

DANS LA MÊME COLLECTION

Amstrad jeux d'action, P. Monsaut
Amstrad premiers programmes, R. Zaks
Amstrad 56 programmes, S.R. Trost
Amstrad guide du BASIC et de l'AMSDOS, J.-L. Gréco/M. Laurent
Amstrad exploré, J. Braga
Amstrad programmation en assembleur, G. Fagot-Barraly
Amstrad guide du graphisme, J. Wynford
Amstrad CP/M 2.2, A. d'Hardancourt
Amstrad astrologie, numérologie, biorythmes, P. Bourgault
Amstrad graphisme en trois dimensions, T. Lachant-Robert
Amstrad Multiplan, Amstrad
Amstrad CP/M plus, A. d'Hardancourt
Amstrad Astrocalc, G. Blanc/P. Destrebecq
Amstrad gagnez aux courses, J.-C. Despoine
Amstrad créer de nouvelles instructions, J.-C. Despoine
Amstrad Locoscript, B. Le Dû
Amstrad Logo, A. d'Hardancourt *(à paraître)*
Amstrad programmes en langage machine, S. Webb *(à paraître)*
Amstrad guide du DOS, Amstrad *(à paraître)*
Amstrad introduction à la programmation en assembleur du Z80, A. d'Hardancourt *(à paraître)*
Amstrad systèmes d'exploitation, Amstrad *(à paraître)*
Amstrad programmes scientifiques, Y. Muggianu/M. Lamarche/P.-M. Beaufils *(à paraître)*
Amstrad routines en assembleur, J.-C. Despoine *(à paraître)*
Amstrad jeux en assembleur, E. Ravis *(à paraître)*
Amstrad mieux programmer en assembleur, T. Lachant-Robert *(à paraître)*

C. VIVIER / J. JACOB

AMSTRAD

MISE AU POINT DES PROGRAMMES BASIC



Paris • Berkeley • Düsseldorf

Sybex n'est lié à aucun constructeur.

ISBN 2-7361-0166-8

S O M M A I R E

1

RENSEIGNEMENTS SUR LE MICROPROCESSEUR Z80

Un ordinateur est composé de différents éléments dont le point central est le microprocesseur. Dans le cas de l'Amstrad, il s'agit du Z80.

Le microprocesseur est capable d'effectuer certaines opérations logiques ou arithmétiques ainsi que d'échanger des informations avec les autres éléments. Nous parlons actuellement des possibilités données aux circuits électroniques qui le composent et non des possibilités que l'on peut lui donner par des logiciels.

Nous n'entrerons pas dans le détail du fonctionnement du Z80, nous indiquerons seulement le rôle de quelques-uns de ses composants.

*

*   *

Dans un ordinateur, il existe les types de mémoires suivants :

- Les ROM *(Read Only Memory).* Ces mémoires sont accessibles uniquement en lecture ; il n'est pas possible d'y écrire, sauf au moment de leur réalisation. Ce sont les mémoires "mortes" de l'ordinateur.
- Les RAM, *Random Access Memory.* Ces mémoires sont accessibles en lecture et en écriture. Ce sont les mémoires "vives" de l'ordinateur.
- Les mémoires de travail du microprocesseur sont appelées *registres,* leur particularité par rapport aux ROM et aux RAM est d'avoir un temps d'accès plus court ; certaines sont dotées d'un nom particulier lorsqu'elles ont une utilité spécifique.
- L'ALU *(Arithmetical Logical Unit),* cœur du microprocesseur, doté de la possibilité d'effectuer des opérations arithmétiques et logiques, comme son nom l'indique.

FONCTIONNEMENT DU Z80

Examinons maintenant les registres du Z80 :

- L'*accumulateur,* noté A : ce registre est associé à l'ALU pour lui permettre de travailler.

- Les *registres universels,* qui peuvent avoir plusieurs utilisations ; ils ne sont pas affectés à priori à une tâche particulière.
- Les *registres d'adresses,* notés BC / DE / HL, qui sont des registres doubles : deux registres, indépendants physiquement, sont associés dans leur désignation pour la gestion des adresses ; le poids faible d'une adresse est contenu dans le premier registre, le poids fort dans le second. La notion de poids faible et de poids fort sera développée ultérieurement au paragraphe Écriture d'une adresse.
- Le *compteur ordinal,* noté PC, qui contient l'adresse de la prochaine instruction à exécuter.
- Le *pointeur de pile,* noté SP, qui permettra de gérer les piles.
- Le *registre d'index,* noté IX, qui permettra de faire de l'adressage indexé.

De plus, pour accélérer certains travaux, le Z80 possède deux jeux indépendants de certains des registres énumérés ci-dessus.

Revenons aux registres d'adresses. Nous avons dit que ces registres étaient doubles ; cette particularité permet de gérer les adresses plus rapidement. Pour expliquer leur fonctionnement, commençons tout d'abord par quelques rappels sur les systèmes de numération.

NUMÉRATION DÉCIMALE, BINAIRE ET HEXADÉCIMALE

Lorsque l'on écrit le nombre 627 en numération décimale, la traduction arithmétique est :

$$6*10^2+2*10^1+7*10^0$$

10 est la base de la numération que l'on élève à la puissance 0, puis 1, puis 2, etc. (rappelons que 10^0 est égal à 1).

Le nombre 1011 en numération décimale se traduit par :

$$1*10^3+0*10^2+1*10^1+1*2^0=1011$$

En numération binaire, il se traduit par :

$$1*2^3+0*2^2+1*2^1+1*2^0$$

soit la valeur 14 en numération décimale.

En numération hexadécimale, il se traduit par

$$1*16^3+0*16^2+1*16^1+1*16^0$$

soit la valeur 1554 en numération décimale.

*

* *

Un octet est un mot de huit bits ; chaque bit peut prendre la valeur 0 ou 1 ; dans un octet, les nombres sont représentés en numération binaire. Le plus petit est 00000000 ; le plus grand est 11111111, ce qui nous donne en valeur décimale 0 pour le plus petit et 255 pour le plus grand.

$$1*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+1*2^0=255$$

La numération hexadécimale est souvent utilisée lorsque l'on travaille directement sur le contenu d'un octet, car elle permet d'écrire tous les nombres possibles qu'il peut contenir avec seulement deux chiffres ou lettres.

La succession des chiffres en numération décimale est :

0,1,2,3,4,5,6,7,8,9

La succession des chiffres en numération hexadécimale est :

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

A a la valeur 10, B la valeur 11, C la valeur 12, D la valeur 13, E la valeur 14, F la valeur 15 en numération décimale.

Le nombre 255 en numération décimale s'écrit FF en numération hexadécimale ; c'est le plus grand nombre que l'on puisse écrire dans un octet.

$$F*16^1+F*16^0=15*16+15=255$$

ÉCRITURE D'UNE ADRESSE

Le microprocesseur Z80 est capable de travailler avec des adresses allant de 0 à 65535. Nous venons de voir ci-dessus que, dans un octet, le nombre le plus grand que l'on pouvait représenter était 255 ; par conséquent, il sera nécessaire d'utiliser deux octets pour pouvoir représenter des nombres jusqu'à 65535. Ces deux octets sont utilisés de la manière suivante : l'adresse sera égale à la valeur contenue dans le premier octet augmentée de la valeur contenue dans le second octet multipliée par 256.

EXEMPLE

L'adresse 28712 sera représentée par les deux octets suivants :

00101000   01110000

L'octet 00101000 vaut 40 en décimal, l'octet 01110000 vaut 112 en décimal ; on a bien en décimal 40+256*112=28712

Le premier octet est dit contenir le poids faible, le deuxième octet le poids fort.

Il est d'usage d'appeler l'ensemble des mémoires qui ont la même valeur dans l'octet de poids fort, une *page* ; les mémoires dont l'adresse va de 28160 à 28415 ont comme poids fort 112, ce sont les mémoires de la page 112.

2

LES MÉMOIRES DE L'AMSTRAD

Dans sa version de base, l'Amstrad possède une mémoire RAM de 65 536 octets (soit 64K ; 1K = 1 024 octets), mais toute cette mémoire n'est pas disponible pour l'utilisateur. Un certain nombre de mémoires est réservé pour le fonctionnement du microprocesseur, de l'*interpréteur* BASIC et les affichages à l'écran. Nous allons donner la carte des mémoires ainsi que quelques adresses remarquables utilisées par le microprocesseur ou l'interpréteur.

CARTOGRAPHIE DES MÉMOIRES DE L'AMSTRAD

Les adresses des RAM vont de 0 à 65535.

- Les mémoires d'adresses 0 à 366 (page 0 et début page 1) sont réservées pour le fonctionnement de l'AMSDOS et du CP/M.
- Les mémoires d'adresses 367 à 42619 sont libres pour le programme BASIC et ses variables associées.
- Les mémoires d'adresses 42620 à 49151 sont réservées au fonctionnement du microprocesseur et de l'interpréteur.
- Les mémoires d'adresses 49152 à 65535 sont réservées pour l'affichage écran.

La cartographie de la mémoire se schématise de la manière suivante :

ADRESSE	UTILISATION	APPELLATION
65535		
	Mémoires écran	
49152		
	Réservé interpréteur et microprocesseur	
42620		HIMEM
	Mémoires disponibles pour le BASIC	
367		TXTTAB
	Réservé AMSDOS et CP/M	
0		

L'adresse de début de la zone libre pour le programme BASIC s'appelle TXTTAB ; cette adresse est stockée dans les mémoires réservées à l'interpréteur, soit 44673 (poids faible) et 44674 (poids fort).

L'adresse de la fin de zone libre pour le programme BASIC s'appelle le HIMEM ; cette adresse est stockée dans les mémoires réservées à l'interpréteur, soit 45199 (poids faible) et 45200 (poids fort).

ADRESSES REMARQUABLES DE LA ZONE MÉMOIRE RÉSERVÉE A L'INTERPRÉTEUR

Les adresses remarquables que nous allons citer sont toutes implantées dans la zone réservée à l'interpréteur, nous les donnerons dans l'ordre poids faible, poids fort.

- Nous venons de voir que l'adresse de début de la zone disponible pour le programme BASIC s'appelle TXTTAB ; cette adresse est stockée dans les mémoires 44673 et 44674. L'adresse de la mémoire la plus haute (l'adresse la plus forte disponible) s'appelle le HIMEM ; elle est stockée dans les mémoires d'adresses 45199 et 45200. Elle peut être différente de la plus haute mémoire existante, caractéristique que nous utiliserons ultérieurement.

- L'adresse de fin de programme BASIC s'appelle le LOMEM ; elle est stockée dans les mémoires d'adresses 45675 et 45676.

- Les variables numériques non indicées et l'adresse de stockage des variables alphanumériques non indicées sont stockées dans une zone de mémoire dont l'adresse de début est le LOMEM et l'adresse de fin le ARYTAB. ARYTAB est stocké dans les mémoires d'adresses 44679 et 44680.

- Les variables numériques indicées et les adresses de stockage des variables alphanumériques indicées sont stockées dans une zone dont l'adresse de début est ARYTAB et l'adresse de fin STREND. STREND est stocké dans les mémoires d'adresses 44681 et 44682.

- Les variables alphanumériques, indicées ou non, sont stockées dans une zone dont l'adresse de début est HIMEM et dont l'adresse de fin est FRETOP. FRETOP est stocké dans les mémoires d'adresses 45197 et 45198. Il est à remarquer que pour cette zone l'adresse de début est supérieure à l'adresse de fin, à l'inverse des autres zones. Nous en donnerons l'explication au Chapitre 3.

- La zone allant de FRETOP à STREND est libre.

Il faut bien retenir de ce schéma que :

1. Le LOMEM s'élève au fur et à mesure que le programme s'allonge.
2. La zone des variables qui s'étend de LOMEM à STREND est d'autant plus importante que le nombre ou la dimension des variables sont importants.
3. La zone supérieure qui va de HIMEM à FRETOP contient toutes les variables alphanumériques du programme (par variables, il faut entendre tous les messages introduits par l'utilisateur, par exemple des INPUT R$ ou des concaténations telles que A$ = B$ + C$.
4. Dès que la zone libre deviendra nulle, le programme n'aura plus de place pour stocker les variables, ce qui déclenchera le message d'erreur ''OUT OF MEMORY'' et le programme sera arrêté dans son exécution. Il est donc nécessaire de limiter l'encombrement d'un programme et le nombre de variables utilisées pour travailler correctement dans l'espace mémoire d'un micro-ordinateur.

*

* *

Pour la zone utilisable par le BASIC, nous avons la carte suivante :

Utilisation	Nom	Adresses
	HIMEM	45199 45200
Zone de stockage des variables alphanumériques indicées ou non		
	FRETOP	45197 45198
Zone libre		
	STREND	44681 44682
Zone de stockage des variables numériques indicées Adresses de stockage des variables alphanumériques indicées		
	ARYTAB	44679 44680
Zone de stockage des variables numériques Adresses de stockage des variables alphanumériques non indicées		
	LOMEM	44675 44676
Programme BASIC		
	TXTTAB	44673 44674

Nous utiliserons certaines de ces adresses dans les chapitres suivants.

3

CODAGE DES INFORMATIONS D'UN PROGRAMME BASIC

La première étape nécessaire pour pouvoir explorer un programme est de connaître la manière dont il est stocké dans les mémoires, ce que nous allons examiner ; puis nous étudierons l'implantation et le codage des variables utilisées par le programme.

CODAGE DES ORDRES

Nous avons vu au Chapitre 2 que les mémoires d'adresses 44673 et 44674 contenaient l'adresse du TXTTAB, c'est-à-dire l'adresse à partir de laquelle les instructions du BASIC vont être enregistrées. Nous allons commencer par interroger ces mémoires pour connaître le TXTTAB de l'Amstrad. En ordre à exécution immédiate, tapons :

```
CLS <Return>
PRINT PEEK(44673)+256*PEEK(44674) <Return>
```

Nous obtenons le résultat suivant à l'écran :

```
367
```

Maintenant que nous savons où sont implantées les instructions BASIC, nous allons écrire un programme qui va s'auto-explorer ; nous allons séparer ce programme en deux parties : celle que nous explorerons au début et celle qui réalisera l'exploration.

Ce programme étant du BASIC classique, nous ne signalons plus les RETURN en fin de ligne.

NEW

L'ordre NEW permet de libérer la mémoire de l'ordinateur pour taper un nouveau programme sans le mélanger avec un programme introduit précédemment. Nous verrons ultérieurement l'effet exact de l'ordre NEW sur les mémoires.

```
10 REM CH3-P1
20 PRINT "BONJOUR"
30 A%=2 : B%=145 : C%=758 :D%=-456
40 E=2 : EA=45 : ASR=78.5 :ASDF=-456.2
50 X=&1AEF : Y=&X10100010
60 A$="HELLO"
70 A=PEEK(44675)+256*PEEK(44676)
80 FOR J=367 TO A STEP 30:CLS
90 FOR I=J TO J+28 STEP 2
100 PRINT I;PEEK(I),I+1;PEEK(I+1):NEXT I
110 PRINT:INPUT R$: IF LEFT$(R$,1)<>"C" THEN 140
120 FOR I=J TO J+28 STEP 2:PRINT #8,I;PEEK(I),I+1;PEEK(I+1):NEXT I
130 PRINT #8,:PRINT#8,
140 NEXT J
150 END
```

Le principe de ce programme est d'explorer les mémoires où sont implantées les instructions BASIC. On affiche les codes ASCII trouvés à l'écran par série de 30, puis une nouvelle série de 30 et ainsi de suite.

20/60

Différents types de variables, pour examiner la manière dont elles sont représentées en mémoire.

90/100 et 120

L'affichage, par série de 30, est réalisé par les deux boucles FOR/NEXT en I.

80/140

La boucle FOR/NEXT en J permet l'exploration complète du programme, puisqu'elle commence à partir du TXTTAB, soit 367, jusqu'au LOMEM (fin du programme), soit PEEK(44675)+256*PEEK(44676).

100

Cette instruction affiche le numéro de la mémoire explorée et le code ASCII trouvé (deux fois par ligne).

110

Cette instruction arrête l'exécution jusqu'à ce que l'on frappe la touche RETURN ; si l'on frappe C suivi de RETURN, on déclenche l'impression sur imprimante (instruction valable pour les lecteurs qui en possèdent une). L'ordre INPUT est précédé de PRINT, qui met le curseur à la ligne suivante.

Le BASIC est implanté à partir du TXTTAB et nous allons trouver successivement : la longueur de l'instruction (nombre d'octets occupés par l'instruction en mémoire), l'étiquette de l'instruction, la succession d'ordres BASIC de cette étiquette, puis l'instruction suivante et ainsi de suite.

Les ordres BASIC tels que PRINT, CLS, PEEK... ne sont pas stockés sous leur forme littérale, ce qui consommerait trop d'octets (un octet par lettre soit cinq pour l'ordre PRINT) ; ils sont représentés par un ou deux codes ASCII supérieurs à 127, ce qui facilite le travail de

l'interpréteur pour la traduction : en effet, les octets des tokens commencent alors par un bit égal à 1.

Exécutons maintenant ce programme par RUN. On voit apparaître à l'écran :

```
367     0       368     13
369     0       370     10
371     0       372     197
373     32      374     67
375     72      376     51
377     45      378     80
379     49      380     0
381     16      382     0
383     20      384     0
385     191     386     32
387     34      388     66
389     79      390     78
391     74      392     79
393     85      394     82
395     34      396     0
```

Voici les seize écrans que vous devez obtenir (seule la partie intéressante est représentée).

```
            ECRAN 1                               ECRAN 2

****************************** ******************************
*                            * *                            *
$ 367     0       368     13 * * 397     43      398     0  *
* 369     0       370     10 * * 399     30      400     0  *
* 317     0       372    197 * * 401     2       402     5  *
* 373     32      374     67 * * 403     0       404    193 *
* 375     72      376     51 * * 405     239     406     16 *
* 377     45      378     80 * * 407     32      408     1  *
* 379     49      380     0  * * 409     32      410     2  *
* 381     16      382     0  * * 411     11      412     0  *
* 383     20      384     0  * * 413     194     414    239 *
* 385     191     386     32 * * 415     25      416    145 *
* 387     34      388     66 * * 417     32      418     1  *
* 389     79      390     78 * * 419     32      420     2  *
* 391     74      392     79 * * 421     17      422     0  *
* 393     85      394     82 * * 423     195     424    239 *
* 395     34      396     0  * * 425     26      426    246 *
*                            * *                            *
****************************** ******************************
```

ECRAN 3

427	2	428	32
429	1	430	2
431	23	432	0
433	196	434	239
435	245	436	26
437	200	438	1
439	0	440	55
441	0	442	50
443	0	444	13
445	29	446	0
447	197	448	239
449	16	450	32
451	1	452	32
453	13	454	39
455	0	456	69

ECRAN 4

457	193	458	239
459	25	460	45
461	32	462	1
463	32	464	13
465	50	466	0
467	65	468	83
469	210	470	239
471	31	472	0
473	0	474	0
475	29	476	135
477	32	478	1
479	13	480	62
481	0	482	65
483	83	484	68
485	198	486	239

ECRAN 5

487	245	488	31
489	154	490	153
491	25	492	100
493	137	494	0
495	24	496	0
497	50	498	0
499	13	500	71
501	0	502	216
503	239	504	28
505	239	506	26
507	32	508	1
509	32	510	13
511	80	512	0
513	217	514	239
515	27	516	162

ECRAN 6

517	0	518	0
519	17	520	0
521	60	522	0
523	3	524	89
525	0	526	193
527	239	528	34
529	72	530	69
531	76	532	76
533	79	534	34
535	0	536	35
537	0	538	70
539	0	540	13
541	96	542	0
543	193	544	239
545	255	546	18

ECRAN 7

547	40	548	31
549	0	550	0
551	131	552	46
553	144	554	41
555	244	556	26
557	0	558	1
559	246	560	255
561	18	562	40
563	31	564	0
565	0	566	132
567	46	568	144
569	41	570	0
571	29	572	0
573	80	574	0
575	158	576	32

ECRAN 8

577	13	578	105
579	0	580	202
581	239	582	26
583	111	584	1
585	32	586	236
587	32	588	13
589	96	590	0
591	193	592	32
593	230	594	32
595	25	596	30
597	1	598	138
599	0	600	30
601	0	602	90
603	0	604	158
605	32	606	13

ECRAN 9				ECRAN 10			
607	114	608	0	637	114	638	0
609	201	610	239	639	201	640	59
611	13	612	105	641	255	642	18
613	0	614	202	643	40	644	13
615	32	616	236	645	114	646	0
617	32	618	13	647	201	648	41
619	105	620	0	649	44	650	13
621	202	622	244	651	114	652	0
623	25	624	28	653	201	654	244
625	32	626	230	655	15	656	59
627	32	628	16	657	255	658	18
629	0	630	45	659	40	660	13
631	0	632	100	661	114	662	0
633	0	634	191	663	201	664	244
635	32	636	13	665	15	666	41

ECRAN 11				ECRAN 12			
667	1	668	176	697	210	698	44
669	32	670	13	699	15	700	41
$ 671	114	672	0	701	242	702	34
673	201	674	0	703	67	704	34
675	40	676	0	705	32	706	235
677	110	678	0	707	32	708	29
679	191	680	1	709	35	710	3
681	163	682	32	711	32	712	32
683	3	684	123	713	32	714	0
685	0	686	210	715	74	716	0
687	1	688	32	717	120	718	0
689	161	690	32	719	158	720	32
691	255	692	117	721	13	722	114
693	40	694	3	723	0	724	201
695	123	696	0	725	239	726	13

ECRAN 13				ECRAN 14			
727	105	728	0	757	40	758	13
729	202	730	32	759	114	760	0
731	236	732	32	761	201	762	41
733	13	734	105	763	44	764	13
735	0	736	202	765	114	766	0
737	244	738	25	767	201	768	244
739	28	740	32	769	15	770	59
741	230	742	32	771	255	772	18
743	16	744	1	773	40	774	13
745	191	746	32	775	114	776	0
747	35	748	22	777	201	778	244
749	44	750	13	779	15	780	41
751	114	752	0	781	1	782	176
753	201	754	59	783	32	784	13
755	255	756	18	785	114	786	0

```
          ECRAN 15                                   ECRAN 16

*************************************************************
*                                    * *                    *
* 787      201      788      0      * * 817      150      818      0    *
* 789      15       790      0      * * 819      191      820      32   *
* 791      130      792      0      * * 821      13       822      0    *
* 793      191      794      32     * * 823      0        824      193  *
* 795      35       796      22     * * 825      0        826      10   *
* 797      44       798      1      * * 827      0        828      160  *
* 799      191      800      35     * * 829      0        830      160  *
* 801      22       802      44     * * 831      32       832      29   *
* 803      0        804      11     * * 833      67       834      3    *
* 805      0        806      140    * * 835      0        836      6    *
* 807      0        808      176    * * 837      0        838      170  *
* 809      32       810      13     * * 839      0        840      152  *
* 811      105      812      0      * * 841      0        842      0    *
* 813      202      814      0      * * 843      0        844      0    *
* 815      11       816      0      * * 845      0        846      193  *
*                                    * *                    *
*************************************************************
```

Examinons la signification de ces codes ASCII :

Mémoire	Code	Signification
367	0	Début du programme BASIC.
368	13	13+256*0=13 ; longueur de l'instruction
369	0	BASIC qui suit.
370	10	10+256*0=10 ; étiquette de
371	0	l'instruction BASIC considérée.
372	197	Le code ASCII 197 correspond à l'ordre BASIC REM : c'est un token.
373	32	Le code ASCII 32 correspond à un espace.
374	67	Code ASCII de C.
375	72	Code ASCII de H.
376	51	Code ASCII de 3.
377	45	Code ASCII de –.
378	80	Code ASCII de P.
379	49	Code ASCII de 1.
380	0	Fin de l'instruction BASIC n° 10 ; début de la suivante.
381	16	16+256*0=16 ; longueur de l'instruction
382	0	BASIC qui suit.
383	20	20+256*0=20 ; étiquette de
384	0	l'instruction BASIC considérée.

385	191	Le code ASCII 191 correspond à l'ordre BASIC PRINT : c'est un token.
386	32	Code ASCII de espace.
387	34	Code ASCII de ".
388	66	Code ASCII de B.
389	79	Code ASCII de O.
390	78	Code ASCII de N.
391	74	Code ASCII de J.
392	79	Code ASCII de O.
393	85	Code ASCII de U.
394	82	Code ASCII de R.
395	34	Code ASCII de ".
396	0	Fin de l'instruction BASIC n° 20 ; début de la suivante.

Nous venons donc de constater les faits suivants :

- Un programme BASIC commence à l'adresse 367.
- Un programme BASIC est précédé d'un 0.
- Toutes les instructions BASIC sont séparées les unes des autres par un 0 (nous verrons par la suite que le 0 est utilisé aussi pour d'autres besoins).
- Une instruction BASIC commence par la longueur occupée par cette instruction en mémoire ; cette longueur est stockée dans deux mémoires.
- L'ordre REM est stocké sous forme d'un seul code ASCII, de même l'ordre PRINT ainsi que les autres ordres BASIC ; ce code ASCII a une valeur supérieure à 127, il est habituellement appelé *token*. La liste des tokens de l'Amstrad est donnée en Annexe C.

Nous allons continuer l'interprétation des instructions de notre programme en regroupant les codes ASCII entre deux 0 de fin d'instruction.

A partir de la mémoire 397, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
30 A%=2 : B%=145 : C%=758 : D%=-456
```

43-0	43+256*0=43 ; longueur de l'instruction.
30-0	30+256*0=30 ; étiquette de l'instruction.
2	Indicateur de variable entière.
5-0	La valeur de cette variable est stockée à partir de l'adresse égale à celle du LOMEM+5+0*256=LOMEM+5.
193	193 est le code ASCII de A (soit 65) augmenté de 128 ; c'est la variable A% et A est le dernier caractère de cette variable.
239	Token de =.
16	Valeur numérique codée, 16 correspond à la valeur 3.
32	Code ASCII de espace.
1	Séparation de deux ordres à l'intérieur d'une instruction ; c'est la traduction de : de séparation.
32	Code ASCII de espace.
2-11-0	Variable entière dont la valeur est stockée à LOMEM+11.
194	Nom de la variable B% (code ASCII de B+128).
239	Token de =.
25	Constante codée sur un octet.
145	Valeur de la constante : 145.
32-1-32	Espace : espace.
2-17-0	Variable entière dont la valeur est stockée à LOMEM+17.
195	Code ASCII de C+128, variable C%.
239	Token de =.
26	Constante codée sur deux octets.
246-2	Valeur de cette constante, soit 246+2*256=758.
32-1	Espace : .
2-23-0	Variable entière dont la valeur est stockée à LOMEM+23.
196	Code ASCII de D+128, variable D%.
239	Token de =.
245	Token de −.
26-200-1	Constante codée sur deux octets, soit 200+1*256=456.

Le codage des variables et des constantes mérite quelques explications complémentaires avant de continuer.

Codage des variables

On trouve tout d'abord un octet pour indiquer le type de variable ; cet octet est égal à *2* pour une *variable entière,* à *3* pour une *variable alphanumérique,* à *13* pour une *variable en virgule flottante.*

Puis les deux octets suivants contiennent l'adresse, dans la zone LOMEM/ARYTAB ou ARYTAB/STREND, de la valeur de cette variable. Avant de faire RUN, ces deux octets sont bien sûr à 0, car l'interpréteur ne sait pas encore où seront stockées les valeurs ; au fur et à mesure que l'interpréteur rencontre les variables pendant l'exécution du programme, il attribue un emplacement pour la valeur de la variable dans la zone LOMEM/ARYTAB ou ARYTAB/STREND et il note l'adresse dans ces deux octets. Cette particularité du BASIC de l'Amstrad permet de diminuer les temps de calcul.

Enfin on trouve les codes ASCII représentant les caractères de la variable ; la longueur de chaîne n'étant pas imposée, on augmente le code ASCII du dernier caractère de 128 pour indiquer la fin de la chaîne.

Codage des constantes

La procédure de codage des variables augmente la rapidité de calcul, mais elle augmente aussi la taille du programme en mémoire. Pour les constantes, l'Amstrad utilise aussi une technique qui augmente la rapidité de calcul mais qui, cette fois, diminue l'encombrement mémoire.

Si la constante ne comporte qu'un chiffre, celui-ci est codé directement avec la convention suivante :

Octet = 14 si la constante est égale à 0
Octet = 15 si la constante est égale à 1
Octet = 16 si la constante est égale à 2
Octet = 17 si la constante est égale à 3
Octet = 18 si la constante est égale à 4
Octet = 19 si la constante est égale à 5
Octet = 20 si la constante est égale à 6

Octet = 21 si la constante est égale à 7
Octet = 22 si la constante est égale à 8
Octet = 23 si la constante est égale à 9

Si la constante est un entier inférieur à 255, on code cette valeur directement dans un octet et on le fait précéder d'un octet égal à *25.*

Si la constante est un entier supérieur à 255 et inférieur à 65535, on code cette valeur sur deux octets suivant la technique habituelle : premier octet augmenté de 256 fois le second, et on les fait précéder par un octet égal à *26.*

Si la constante est un nombre en hexadécimal, elle commence par & et sa valeur est inférieure à 65535 : on la code donc comme ci-dessus et on fait précéder les deux octets par un octet égal à *28.*

Si la constante est un nombre en binaire, elle commence par &X et sa valeur est inférieure à 65535 ; on la code comme ci-dessus et on fait précéder les deux octets par un octet égal à *27.*

Si la constante représente une adresse (par exemple dans un GOTO), elle est inférieure à 65535 ; on peut donc la coder sur deux octets que l'on fait précéder de *30,* et cela avant l'exécution du programme. Mais lorsque l'on exécute le programme, l'interpréteur trouve au fur et à mesure de ses besoins les adresses réelles de branchement ; il remplace alors la valeur des étiquettes par l'adresse absolue, qui se code toujours sur deux octets qu'il fait précéder par un octet égal à *29,* pour reconnaître une étiquette d'une adresse relative.

Enfin, si l'on a une constante ne répondant à aucune des classes précédentes, on met un premier octet égal à 31 suivi par cinq octets qui définissent sa valeur, en utilisant la méthode décrite à la section suivante (codage de la valeur des variables en virgule flottante).

Les valeurs négatives s'obtiennent par le token de "–" en tête de la constante.

Nous pouvons maintenant reprendre notre exploration.

A partir de la mémoire 440, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
40 E=2 : EA=45 : ASR=78.5 : ASDF=456.2
```

55-0 55+256*0=55 ; longueur de l'instruction.
40-0 40+256*0=40 ; étiquette de l'instruction.

13-29-0-197	197=128+69 ; 69 est le code ASCII de E : variable E en virgule flottante stockée en LOMEM+29.
239	Token de =.
16	Constante égale à 2.
32-1-32	Séparation espace : espace.
13-39-0	Variable en virgule flottante stockée en LOMEM+39.
69-193	Codes ASCII de E et A+128 ; variable EA.
239	Token de =.
25-45	Constante sur un octet, valeur 45.
32-1-32	Séparation espace : espace.
13-50-0	Variable en virgule flottante stockée en LOMEM+50.
65-83-210	Code ASCII de A, S et R+128 ; variable ASR.
239	Token de =.
31	Constante en virgule flottante.
0-0-0-29-135	Valeur (1+29/128)*2^(135−129)=78.5.
32-1	Séparation espace :.
13-62-0	Variable en virgule flottante stockée en LOMEM+62.
65-83-68-198	Codes ASCII de A, S, D et F+128 ; variable ASDF.
239	Token de =.
245	Token de −.
31	Variable en virgule flottante.
154-153-25 100-137	(1+100/128+25/32768+153/8388608+ 154/2147483648)*2^(137−129)= 456.19999.

A partir de la mémoire 495, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
50 X=&1AEF : Y=&X10100010
```

24-0	24+256*0=24 ; longueur de l'instruction.
50-0	50+256*0=50 ; étiquette de l'instruction.
13-71-0	Variable en virgule flottante stockée en LOMEM+71.

216 Code ASCII de X, +128, variable X.

239 Token de =.

28-239-26 Constante en hexadécimal, codée sur 2 octets ; valeur 239+26*256=6895 ou &1AEF.

32-1-32 Séparation espace : espace.

13-80-0 Variable en virgule flottante stockée en LOMEM+80.

217 Code ASCII de Y+128 ; variable Y.

239 Token de =.

27-162-0 Constante en binaire, codée sur 2 octets ; valeur 162 ou &X10100010.

A partir de la mémoire 519, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
60 A$="HELLO"
```

17-0 17+256*0=17 ; longueur de l'instruction.

60-0 60+256*0=60 ; étiquette de l'instruction.

3-89-0 Variable alphanumérique stockée en LOMEM+89.

193 Code ASCII de "A"+128, variable A.

239 Token de =.

34 Code ASCII de ".

72 Code ASCII de H.

69 Code ASCII de E.

76 Code ASCII de L.

76 Code ASCII de L.

79 Code ASCII de O.

34 Code ASCII de ".

A partir de la mémoire 536, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
70 A=PEEK(44675)+256*PEEK(44676)
```

35-0 35+256*0=35 ; longueur de l'instruction.

70-0 70+256*0=70 ; étiquette de l'instruction.

13-96-0 Variable en virgule flottante stockée en LOMEM+96.

193 Code ASCII de A, +128, variable A.
239 Token de =.
255-18 Token de PEEK.
40 Code ASCII de (.
31 Constante en virgule flottante codée sur 5 octets.
0-0-131 (1+46/128+131/32768)*2^(144−129)=
46-144 44675.
41 Code ASCII de).
244 Token de +.
26-0-1 Constante codée sur 2 octets ; valeur 0+256*1=256.
246 Token de *.
255-18 Token de PEEK.
40 Code ASCII de (.
31 Constante en virgule flottante codée sur 5 octets.
0-0-132 (1+46/128+132/32768)*2^(144−129)=
46-144 44676.
41 Code ASCII de).

A partir de la mémoire 571, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
80 FOR J=367 TO A STEP 30:CLS
```

29-0 29+256*0=29 ; longueur de l'instruction.
80-0 80+256*0=80 ; étiquette de l'instruction.
158 Token de FOR.
32 Code ASCII de espace.
13-105-0 Variable en virgule flottante stockée en LOMEM+105.
202 Code ASCII de J, +128 ; variable J.
239 Token de =.
26-111-1 Constante codée sur 2 octets ; valeur 111+256*1=367.
32 Code ASCII de espace.
236 Token de TO.
32 Code ASCII de espace.
13-96-0 Variable en virgule flottante stockée en LOMEM+96.
193 Code ASCII de A, +128 ; variable A.

32 Code ASCII de espace.
230 Token de STEP.
32 Code ASCII de espace.
25-30 Constante codée sur un octet, valeur 30.
1 Séparation :.
138 Token de CLS.

A partir de la mémoire 600, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
90 FOR I=J TO J+28 STEP 2
```

30-0 30+256*0=30 ; longueur de l'instruction.
90-0 90+256*0=90 ; étiquette de l'instruction.
158 Token de FOR.
32 Code ASCII de espace.
13-114-0 Variable en virgule flottante stockée en LOMEM+114.
201 Code ASCII de I, +128 ; variable I.
239 Token de =.
13-104-0 Variable en virgule flottante stockée en LOMEM+105.
202 Code ASCII de J, +128 ; variable J.
32 Code ASCII de espace.
236 Token de TO.
32 Code ASCII de espace.
13-105-0 Variable en virgule flottante stockée en LOMEM+105.
202 Code ASCII de J, +128 ; variable J.
244 Token de +.
25-28 Constante codée sur 1 octet, valeur 28.
32 Code ASCII de espace.
230 Token de STEP.
32 Code ASCII de espace.
16 Constante de valeur 2.

A partir de la mémoire 630, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
100 PRINT I;PEEK(I),I+1;PEEK(I+1):NEXT I
```

45-0	45+256*0=45 ; longueur de l'instruction.
100-0	100+256*0=100 ; étiquette de l'instruction.
191	Token de PRINT.
32	Code ASCII de espace.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I +128 ; variable I.
59	Code ASCII du caractère ;.
255-18	Token de PEEK.
40	Code ASCII du caractère (.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I +128 ; variable I.
41	Code ASCII du caractère).
44	Code ASCII du caractère ,.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I +128 ; variable I.
244	Token de +.
15	Constante de valeur 1.
49	Code ASCII du caractère ;.
255-18	Token de PEEK.
40	Code ASCII du caractère (.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I +128 ; variable I.
244	Token de +.
15	Constante de valeur 1.
41	Code ASCII du caractère).
176	Token de NEXT.
32	Code ASCII de espace.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I, +128 ; variable I.

A partir de la mémoire 675, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
110 PRINT:INPUT R$: IF LEFT$(R$,1) < > "C" THEN 140
```

45-0	45+256*0=45 ; longueur de l'instruction.
110-0	110+256*0=110 ; étiquette de l'instruction.
191	Token de PRINT.
1	Séparation :.
163	Token de INPUT.
32	Code ASCII de espace.
3-123-0	Variable alphanumérique stockée en LOMEM+123.
210	Code ASCII de R+128 ; variable R$.
1	Séparation :.
161	Token de IF.
32	Code ASCII de espace.
32	Code ASCII de espace.
255-117	Token de LEFT$.
40	Code ASCII du caractère (.
3-123-0	Variable alphanumérique stockée en LOMEM+123.
210	Code ASCII de R+128 ; variable R$.
44	Code ASCII du caractère ,.
15	Constante de valeur 1.
41	Code ASCII du caractère).
242	Token de < >.
34	Code ASCII du caractère ''.
67	Code ASCII du caractère C.
34	Code ASCII du caractère ''.
32	Code ASCII de espace.
235	Token de THEN.
32	Code ASCII de espace.
29-35-3	Adresse absolue codée sur 2 octets ; valeur 35+256*3=803.

A partir de la mémoire 715, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
120 FOR I=J TO J+28 STEP 2:PRINT #8,I;PEEK(I),I+1;PEEK(I+1):NEXT I
```

74-0	74+256*0=74 ; longueur de l'instruction.
120-0	120+256*0=120 ; étiquette de l'instruction.

158	Token de FOR.
32	Code ASCII de espace.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I+128 ; variable I.
239	Token de =.
13-105-0	Variable en virgule flottante stockée en LOMEM+105.
202	Code ASCII de J+128 ; variable J.
32	Code ASCII de espace.
236	Token de TO.
32	Code ASCII de espace.
13-105-0	Variable en virgule flottante stockée en LOMEM+105.
202	Code ASCII de J+128 ; variable J.
244	Token de +.
25-28	Constante codée sur 1 octet, valeur 28.
32	Code ASCII de espace.
230	Token de STEP.
32	Code ASCII de espace.
16	Constante de valeur 2.
1	Séparation :.
191	Token de PRINT.
32	Code ASCII de espace.
35	Code ASCII de #.
22	Constante de valeur 2.
44	Code ASCII de ,.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I+128 ; variable I.
59	Code ASCII de ;.
255-18	Token de PEEK.
40	Code ASCII de (.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I+128 ; variable I.
41	Code ASCII de).
44	Code ASCII de ,.
13-114-0	Variable en virgule flottante stockée en LOMEM+114.
201	Code ASCII de I+128 ; variable I.

244 Token de +.
15 Constante de valeur 1.
59 Code ASCII de ;.
255-18 Token de PEEK.
40 Code ASCII de (.
13-114-0 Variable en virgule flottante stockée en LOMEM+114.
201 Code ASCII de I+128 ; variable I.
244 Token de +.
15 Constante de valeur 1.
41 Code ASCII de).
1 Séparation :.
176 Token de NEXT.
32 Code ASCII de espace.
13-114-0 Variable en virgule flottante stockée en LOMEM+114.
201 Code ASCII de I+128 ; variable I.

A partir de la mémoire 789, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
130 PRINT #8,:PRINT#8
```

15-0 15+256*0=15 ; longueur de l'instruction.
130-0 29+256*0=130 ; étiquette de l'instruction.
191 Token de PRINT.
32 Code ASCII de espace.
35 Code ASCII de #.
22 Constante de valeur 8.
44 Code ASCII de ,.
1 Séparation :.
191 Token de PRINT.
35 Code ASCII de #.
22 Constante de valeur 8.
44 Code ASCII de ,.

A partir de la mémoire 804, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
140 NEXT J
```

11-0 11+256*0=11 ; longueur de l'instruction.
140-0 140+256*Q=140 ; étiquette de l'instruction.
176 Token de NEXT.
32 Code ASCII de espace.
13-105-0 Variable en virgule flottante stockée en LOMEM+105.
202 Code ASCII de J+128 ; variable J.

A partir de la mémoire 815, nous trouvons les codes ASCII correspondant à l'instruction BASIC :

```
150 END
```

11-0 11+256*0=11 ; longueur de l'instruction.
140-0 140+256*0=140 ; étiquette de l'instruction.
152 Token de END.

Puis nous trouvons quatre codes ASCII égaux à zéro ; le premier correspond à la fin d'instruction, les trois suivants à l'indication de fin de programme BASIC.

La longueur de l'instruction est stockée en début d'instruction pour les raisons suivantes.

Lorsque l'interpréteur BASIC rencontre pour la première fois un GOTO ou un GOSUB suivi d'une étiquette, il ne sait pas à partir de quelle mémoire est implantée l'instruction vers laquelle on demande le branchement. L'interpréteur commence par regarder dans les mémoires 31467 et 31468 le numéro d'étiquette de la première instruction ; si c'est la bonne, il effectue le branchement du programme sur cette instruction. Dans le cas contraire, il regarde dans les mémoires 368 et 369 la longueur de l'instruction ; il calcule l'adresse du début de l'instruction suivante, et il va regarder à cette nouvelle adresse l'étiquette de l'instruction suivante ; si c'est la bonne, il effectue le branchement ; sinon, il calcule l'adresse de l'instruction suivante, et ainsi de suite. Si la longueur de l'instruction n'était pas indiquée, l'interpréteur serait obligé d'explorer toute l'instruction jusqu'à l'étiquette suivante, ce qui lui ferait perdre beaucoup de temps. Le même principe est utilisé pour l'ordre LIST.

*

* *

Une dernière remarque : l'effet de la commande NEW sur l'Amstrad est beaucoup plus radical que sur d'autres ordinateurs. En effet, de nombreux ordinateurs se contentent de remettre les quatre premières mémoires du programme à zéro, ce qui permet de récupérer après quelques PEEK et POKE un programme intempestivement détruit par NEW. Par contre, l'Amstrad remet toutes les mémoires dans leur état initial et le programme est complètement et définitivement effacé.

CODAGE DES VARIABLES

Introduisons le programme suivant :

```
5 REM CH3-P2
10 CLS : A1=0 : A2=0 : A3=0 : A4=0 : A5=0 : I=0 : J=0 : R$="Z"
20 A=256 : BC=2 : D1EF=1.2 : G%=4 : HI%=5 : L$="BONJOUR" : MM$="QWERTY"
30 X=&2AEF : Y=&X10100010
40 LL$=L$ + MM$ : L$=MM$ + "1234"
50 INPUT W$ : INPUT W$
60 DIM Z(2),ZZ(1,2),X%(1),Y$(3)
70 FOR I=0 TO 2 : Z(I)=I : NEXT I
80 FOR I=0 TO 1 : X%(I)=100*I : NEXT I
90 FOR I=0 TO 1 : FOR J=0 TO 2 : ZZ(I,J)=10*I+J : NEXT J : NEXT I
100 FOR I=0 TO 3 : Y$(I)=CHR$(I+65) : NEXT I
110 A1=PEEK(44675)+A*PEEK(44676) : REM  LOMEM
120 A2=PEEK(44679)+A*PEEK(44680) : REM ARYTAB
130 A3=PEEK(44681)+A*PEEK(44682) : REM STREND
140 A4=PEEK(45197)+A*PEEK(45198) : REM FRETOP
150 A5=PEEK(45199)+A*PEEK(45200) : REM  HIMEM
160 CLS
170 FOR J=A1 TO A2 STEP 30
180 PRINT "LOMEM A ARYTAB";A1;"/";A2
190 FOR I=J TO J+28 STEP 2 : IF I>A2 THEN 210
200 PRINT : PRINT I;PEEK(I),I+1;PEEK(I+1);
210 NEXT I
220 PRINT : PRINT : INPUT R$ : IF LEFT$(R$,1)<>"C" THEN 270
230 PRINT #8, : PRINT #8,"LOMEM A ARYTAB";A1;"/";A2
240 FOR I=J TO J+28 STEP 2 : IF I>A2 THEN 260
250 PRINT #8, : PRINT #8,I;PEEK(I),I+1;PEEK(I+1);
260 NEXT I : PRINT #8, : PRINT #8,
270 CLS
280 NEXT J
290 FOR J=A2 TO A3 STEP 30
300 PRINT "ARYTAB A STREND";A2;"/";A3
310 FOR I=J TO J+28 STEP 2 : IF I>A3 THEN 330
320 PRINT : PRINT I;PEEK(I),I+1;PEEK(I+1);
330 NEXT I
340 PRINT : PRINT : INPUT R$ : IF LEFT$(R$,1)<>"C" THEN 390
350 PRINT #8, : PRINT #8,"ARYTAB A STREND";A2;"/";A3
360 FOR I=J TO J+28 STEP 2 : IF I>A3 THEN 380
370 PRINT #8, : PRINT #8,I;PEEK(I),I+1;PEEK(I+1);
380 NEXT I : PRINT #8, : PRINT #8,
390 CLS
400 NEXT J
410 FOR J=A5 TO A4 STEP -30
420 PRINT "HIMEM A FRETOP";A5;"/";A4
430 FOR I=J TO J-28 STEP -2
440 PRINT : PRINT I;PEEK(I),I+1;PEEK(I+1);
450 NEXT I
460 PRINT : PRINT : INPUT R$ : IF LEFT$(R$,1)<>"C" THEN 510
470 PRINT #8, : PRINT #8,"HIMEM A FRETOP";A5;"/";A6
```

```
480 FOR I=J TO J-28 STEP -2 : IF I<A4 THEN 500
490 PRINT #8, : PRINT #8,I;PEEK(I),I+1;PEEK(I+1);
500 NEXT I : PRINT #8, : PRINT #8,
510 CLS
520 NEXT J
530 FOR I=A5 TO A4 STEP -1
540 PRINT CHR$(PEEK(I)); : NEXT I
550 PRINT : PRINT : INPUT R$ : IF LEFT$(R$,1)<>"C" THEN 590
560 PRINT #8, : PRINT #8,
570 FOR I=A5 TO A4 STEP -1 : PRINT #8,CHR$(PEEK(I)); : NEXT I
580 PRINT #8
590 END
```

Dans sa première partie, qui va jusqu'à l'ordre 100, nous préparons les mémoires qui vont contenir les variables en leur donnant une valeur. Puis, après avoir calculé les valeurs actuelles de LOMEM, ARYTAB, STREND, FRETOP et HIMEM, nous faisons successivement l'exploration des trois zones LOMEM à ARYTAB, ARYTAB à STREND, HIMEM à FRETOP, tout en les affichant.

10 à 100

Nettoyage de l'écran, puis nous donnons des valeurs à toutes les variables que nous désirons utiliser : soit pour réaliser le programme (variables A1, A2, A3, A4, A5, I, J, A), soit celles qui sont placées à titre d'exemple, cela afin de réserver l'espace mémoire des variables avant de demander les valeurs de ARYTAB et STREND. Nous avons choisi un ou plusieurs exemples de chaque type de variable :

- variables dites en *virgule flottante,*
- variables *entières,*
- variables *alphanumériques,* dimensionnées ou non.

110 à 150

Nous calculons les valeurs actuelles de LOMEM, ARYTAB, STREND, FRETOP et HIMEM.

160 à 280

Exploration et affichage des mémoires comprises entre LOMEM et ARYTAB, avec option d'édition sur imprimante si l'on frappe un C.

290 à 400

Exploration et affichage des mémoires comprises entre ARYTAB et STREND, avec option d'édition sur imprimante si l'on frappe un C.

410 à 520

Exploration et affichage des mémoires comprises entre HIMEM et FRETOP, avec option d'édition sur imprimante si l'on frappe un C.

530 à 580

Boucle permettant la traduction en caractères des codes ASCII trouvés entre HIMEM et FRETOP.

590

Fin du programme.

Lorsque ce programme est introduit, sauvegardez-le puis exécutez-le en donnant comme réponse, pour les deux valeurs de W$: AAA puis BBBBB, et Z pour ne pas avoir d'édition sur l'imprimante ; vous devez obtenir les neuf écrans suivants :

```
              ECRAN 1                                   ECRAN 2

******************************** ********************************
*                              * *                              *
* LOMEN A ARYTAB 2604 / 2771   * * LOMEM A ARYTAB 2604 / 2771   *
*                              * *                              *
*  2604  0          2605  0    * *  2634  21         2635  0    *
*  2606  65         2607  145  * *  2636  65         2637  148  *
*  2608  4          2609  0    * *  2638  4          2639  0    *
*  2610  0          2611  192  * *  2640  0          2641  88   *
*  2612  34         2613  140  * *  2642  38         2643  144  *
*  2614  1          2615  0    * *  2644  31         2645  0    *
*  2616  65         2617  146  * *  2646  65         2647  149  *
*  2618  4          2619  0    * *  2648  4          2649  0    *
*  2620  0          2621  48   * *  2650  0          2651  123  *
*  2622  45         2623  140  * *  2652  38         2653  144  *
*  2624  11         2625  0    * *  2654  0          2655  0    *
*  2626  65         2627  147  * *  2656  201        2657  4    *
*  2628  4          2629  0    * *  2658  0          2659  0    *
*  2630  0          2631  112  * *  2660  64         2661  38   *
*  2632  51         2633  140  * *  2662  140        2663  0    *
*                              * *                              *
******************************** ********************************
```

ECRAN 3

```
******************************
*                            *
* LOMEM A ARYTAB 2604 / 2771 *
*                            *
*  2664  0          2665  202 *
*  2666  4          2667  0   *
*  2668  0          2669  128 *
*  2670  38         2671  140 *
*  2672  0          2673  0   *
*  2674  210        2675  2   *
*  2676  1          2677  86  *
*  2678  166        2679  41  *
*  2680  0          2681  193 *
*  2682  4          2683  0   *
*  2684  0          2685  0   *
*  2686  0          2687  137 *
*  2688  0          2689  0   *
*  2690  66         2691  195 *
*  2692  4          2693  0   *
*                            *
******************************
```

ECRAN 4

```
******************************
*                            *
* LOMEM A ARYTAB 2604 / 2771 *
*                            *
*  2694  0          2695  0   *
*  2696  0          2697  130 *
*  2698  0          2699  0   *
*  2700  68         2701  17  *
*  2702  69         2703  198 *
*  2704  4          2705  154 *
*  2706  153        2707  153 *
*  2708  25         2709  129 *
*  2710    0        2711  0   *
*  2712  199        2713  1   *
*  2714  4          2715  0   *
*  2716  0          2717  0   *
*  2718  72         2719  201 *
*  2720  1          2721  5   *
*  2722  0          2723  0   *
*                            *
******************************
```

ECRAN 5

```
******************************
*                            *
* LOMEM A ARYTAB 2604 / 2771 *
*                            *
*                            *
*  2724  0          2725  204 *
*  2726  2          2727  10  *
*  2728  101        2729  166 *
*  2730  0          2731  0   *
*  2732  77         2733  205 *
*  2734  2          2735  6   *
*  2736  26         2737  2   *
*  2738  0          2739  0   *
*  2740  216        2741  4   *
*  2742  0          2743  0   *
*  2744  188        2745  43  *
*  2746  142        2747  0   *
*  2748  0          2749  217 *
*  2750  4          2751  0   *
*  2752  0          2753  0   *
*                            *
******************************
```

ECRAN 6

```
******************************
*                            *
* LOMEM A ARYTAB 2604 / 2771 *
*                            *
*                            *
*  2754  34         2755  136 *
*  2756  120        2757  0   *
*  2758  76         2759  204 *
*  2760  2          2761  13  *
*  2762  111        2763  166 *
*  2764  0          2765  0   *
*  2766  215        2767  2   *
*  2768  5          2769  93  *
*  2770  166        2771  0   *
*                            *
*                            *
*                            *
*                            *
*                            *
*                            *
*                            *
******************************
```

ECRAN 7

```
******************************
*                            *
* ARYTAB A STREND 2771 / 2871 *
*                            *
*  2771  0          2772  0   *
*  2773  218        2774  4   *
*  2775  18         2776  0   *
*  2777  1          2778  3   *
*  2779  0          2780  0   *
*  2781  0          2782  0   *
*  2783  0          2784  0   *
*  2785  0          2786  0   *
*  2787  0          2788  0   *
*  2789  129        2790  0   *
*  2791  0          2792  0   *
*  2793  0          2794  130 *
*  2795  1          2796  0   *
*  2797  90         2798  218 *
*  2799  4          2800  35  *
*                            *
******************************
```

ECRAN 8

```
******************************
*                            *
* ARYTAB A STREND 2771 / 2871 *
*                            *
*  2801  0          2802  2   *
*  2803  3          2804  0   *
*  2805  2          2806  0   *
*  2807  0          2808  0   *
*  2809  0          2810  0   *
*  2811  0          2812  0   *
*  2813  0          2814  0   *
*  2815  32         2816  132 *
*  2817  0          2818  0   *
*  2819  0          2820  0   *
*  2821  129        2822  0   *
*  2823  0          2824  0   *
*  2825  48         2826  132 *
*  2827  0          2828  0   *
*  2829  0          2830  0   *
*                            *
******************************
```

```
            ECRAN 9                                  ECRAN 10

****************************** ******************************
*                            * *                            *
* ARYTAB A STREND 2771 / 2871 * * ARYTAB A STREND 2771 / 2871 *
*                            * *                            *
*  2831  130        2832  0  * *  2861  166        2862  1  *
*  2833  0          2834  0  * *  2863  91         2864  166 *
*  2835  64         2836  132 * *  2865  1          2866  90 *
*  2837  0          2838  0  * *  2867  166        2868  1  *
*  2839  216        2840  1  * *  2869  89         2870  166 *
*  2841  7          2842  0  * *  2871  91         2872  166 *
*  2843  1          2844  2  * *                            *
*  2845  0          2846  0  * *                            *
*  2847  0          2848  100 * *                            *
*  2849  0          2850  0  * *                            *
*  2851  0          2852  217 * *                            *
*  2853  2          2854  15 * *                            *
*  2855  0          2856  1  * *                            *
*  2857  4          2858  0  * *                            *
*  2859  1          2860  92 * *                            *
*                            * *                            *
****************************** ******************************
```

```
            ECRAN 11                                 ECRAN 12

****************************** ******************************
*                            * *                            *
* HIMEM A FRETOP 42619/42585 * * HIMEM A FRETOP 42619/42585 *
*                            * *                            *
*  42619   89   42620   24   * *  42589   66   42590   66   *
*  42617   82   42618   84   * *  42587   66   42588   65   *
*  42615   87   42616   69   * *  42585   68   42586   67   *
*  42613   82   42614   81   * *  42583   90   42584   90   *
*  42611   79   42612   85   * *  42581   90   42582   90   *
*  42609   78   42610   74   * *  42579   90   42580   90   *
*  42607   66   42608   79   * *  42577   90   42578   90   *
*  42605   51   42606   52   * *  42575   90   42576   90   *
*  42603   49   42604   50   * *  42573   90   42574   90   *
*  42601   84   42602   89   * *  42571   90   42572   90   *
*  42599   69   42600   82   * *  42569    0   42570    0   *
*  42597   81   42598   87   * *  42567    0   42568    0   *
*  42595   65   42596   65   * *  42565    0   42566    0   *
*  42593   66   42594   65   * *  42563    0   42464    0   *
*  42591   66   42592   66   * *  42561    0   42562    0   *
*                            * *                            *
****************************** ******************************
```

```
                         ECRAN 13

           YTREWQRUOJNOB4321YTREWQAAABBBBBABCDZ
```

Commençons tout d'abord par rappeler les informations données au Chapitre 2 ; dans la zone LOMEN/ARYTAB nous allons trouver les variables simples, dans la zone ARYTAB/STREND les variables dimensionnées, dans la zone HIMEM/FRETOP les variables alphanumériques. Les premiers octets des variables, dimensionnées ou non, auront la signification suivante.

Les premiers octets contiendront les codes ASCII représentant le nom de la variable ; le dernier caractère de la variable aura son code

ASCII augmenté de 128, ce qui indiquera à l'interpréteur qu'il s'agit du dernier caractère.

L'octet suivant indiquera le type de variable : il sera égal à *1* pour indiquer une *variable entière,* égal à *2* pour une *variable alphanumérique,* égal à *4* pour une *variable en virgule flottante.* Puis les autres octets auront une signification différente suivant le type des variables .

La valeur des variables simples *entières* est représentée par deux octets. On calcule la valeur suivante : premier octet + 256 fois le deuxième. Si le nombre obtenu est inférieur ou égal à 32767, la variable a pour valeur ce nombre, on a alors un nombre positif ; si le nombre obtenu est supérieur à 32767 on retranche 65536 et c'est ce résultat qui représente la valeur de la variable, on a donc dans ce cas un nombre négatif.

La valeur des variables simples *en virgule flottante* est représentée par cinq octets contenant l'argument et la mantisse du nombre : A1, A2, A3, A4, A5.

La valeur d'un nombre positif sera donnée par :

$$(1 + A4/128 + A3/32768 + A2/8388608 + A1/2147483648) * 2^{A5-129}$$

La valeur d'un nombre négatif par :

$$- (A4/128 + A3/32768 + A2/8388608 + A1/2147483648) * 2^{A5-129}$$

Un nombre positif a son quatrième octet (A4) inférieur à 128 ; un nombre négatif a son quatrième octet (A4) supérieur ou égal à 128, ce qui signifie que le bit de poids fort de l'octet A4 est mis à zéro pour les nombres positifs et à un pour les nombres négatifs, ce qui permet de reconnaître les nombres positifs des nombres négatifs.

EXEMPLE

Le nombre 3 a pour représentation les quatre octets 0-0-0-64-130, soit :

$$(1 + 64/128) * 2^{130-129} = (1 + 0.5)* 2 = 3$$

Les variables *alphanumériques* sont suivies de trois octets : le premier donne la longueur de la chaîne de caractères, les deux suivants l'adresse à laquelle la chaîne est stockée. Suivant la provenance de l'information, la chaîne se trouvera dans le programme BASIC ou dans la zone HIMEM/FRETOP.

Pour les variables dimensionnées, la représentation se complique un peu. Après le type et le nom de la variable, nous trouverons :

- Deux octets indiquant le nombre de mémoires occupées par le tableau.
- Le décompte des octets commençant au premier octet suivant cette information.
- Un octet indiquant le nombre de dimensions.
- Un octet donnant la valeur de la dernière dimension plus une unité ; ce nombre représente le nombre d'éléments relatifs à cette dimension, puisque l'on commence avec l'indice zéro (si l'on dimensionne à 2, il y a trois éléments : indice 0, indice 1, indice 2) ; puis un octet contenant un zéro ; l'avant-dernière dimension, un octet contenant un zéro, et ainsi de suite jusqu'à la première dimension suivie d'un zéro.
- Ensuite on trouve la valeur du premier élément sur *deux octets* pour une *variable entière,* sur *cinq octets* pour une *variable en virgule flottante,* ou la longueur de la chaîne de caractères suivie de son adresse de stockage sur *trois octets* pour une *variable alphanumérique.* Puis le deuxième élément, le troisième, et ainsi de suite. Le rangement est effectué de la manière suivante : on fait d'abord varier le premier indice en gardant tous les autres à zéro, puis on ajoute une unité au deuxième indice et on refait varier le premier indice, et ainsi de suite. Par exemple : pour A(2,3) on trouve successivement A(0,0), A(1,0), A(2,0), A(0,1), A(1,1), A(2,1) A(0,2), A(1,2), A(2,2), A(0,3), A(1,3), A(2,3).

Nous sommes maintenant prêts à examiner ce que vient de nous donner notre programme.

DE LOMEM A ARYTAB

Mémoire	Valeur	Signification
2604	0	Première variable commençant
2605	0	par A.

2606	65	Nom de la variable ; 65 code
2607	145	ASCII de A, 145 code ASCII de 1 augmenté de 128, soit fin du nom de la variable.
2608	4	Variable en virgule flottante.
2609	0	
2610	0	
2611	192	Valeur de la variable dans ce cas :
2612	34	(1 + 34/128 + 192/32768)*2^(140 – 129) =
2613	140	2604.

Regroupons maintenant les octets relatifs à une variable :

2614 à 2623

1-0	Deuxième variable commençant par A.
65-146	Code ASCII de A et de 2 : variable A2.
4	Variable en virgule flottante.
0-0-48-45-140	Valeur de la variable 2771 (1 + 45/128 + 48/32678)*2^(140 – 129).

2624 à 2633

11-0	Troisième variable commençant par A.
65-147	Code ASCII de A et de 3 : variable A3.
4	Variable en virgule flottante.
0-0-112-51-140	Valeur de la variable 2871 : (1 + 51/128 + 112/32678)*2^(140 – 129).

2634 à 2643

21-0	Quatrième variable commençant par A.
65-148	Code ASCII de A et de 4 : variable A4.
4	Variable en virgule flottante.
0-0-88-38-144	Valeur de la variable 42584 : (1 + 38/128 + 88/32768)*2^(144 – 129).

2644 à 2653

31-0	Cinquième variable commençant par A.
65-149	Code ASCII de A et de 5 : variable A5.
4	Variable en virgule flottante.
0-0-123-38-144	Valeur de la variable 42619 : (1+38/128+123/32768)*2^(144−129).

2654 à 2662

0-0	Première variable commençant par I.
201	Code ASCII de I : variable I.
4	Variable en virgule flottante.
0-0-64-38-140	Valeur de la variable 2660 : (1+38/128+64/32768)*2^(140−129).

2663 à 2671

0-0	Première variable commençant par J.
202	Code ASCII de J : variable J.
4	Variable en virgule flottante.
0-0-128-38-140	Valeur de la variable 2664 : (1+38/128+128/32768)*2^(140−129).

2672 à 2678

0-0	Première variable commençant par R.
210	Code ASCII de R : variable R$.
2	Variable alphanumérique.
1	Longueur de la chaîne 1.
86-166	86+256*166=42582, adresse de stockage.

2679 à 2687

41-0	Sixième variable commençant par A.
193	Code ASCII de A : variable A.
4	Variable en virgule flottante.
0-0-0-0-137	Valeur de la variable 256 : 2^(137−129).

2688 à 2697

0-0	Première variable commençant par B.
66-195	Code ASCII de B et de C : variable BC.
4	Variable en virgule flottante.
0-0-0-0-130	Valeur de la variable 2 : 2^(130 – 129).

2698 à 2709

0-0	Première variable commençant par D.
68-17-69-198	Code ASCII de D, 1, E et F : variable D1EF
4	Variable en virgule flottante.
154-153-153-25-129	Valeur de la variable 1.1999999 : (1 + 25/128 + 153/32768 + 153/8388608) * 2^(129 – 129).

2710 à 2715

0-0	Première variable commençant par G.
199	Code ASCII de G : variable G%.
1	Variable entière.
4-0	Valeur de la variable 4.

2716 à 2722

0-0	Première variable commençant par H.
72-201	Code ASCII de H et de I : variable HI%.
1	Variable entière.
5-0	Valeur de la variable 5.

2723 à 2729

0-0	Première variable commençant par L.
204	Code ASCII de L : variable L$.
2	Variable alphanumérique.
10	Longueur variable 10.
101-166	101 + 256 * 266 = 42597, adresse de stockage.

2730 à 2737

0-0	Première variable commençant par M.
77-205	Code ASCII de M et de M : variable MM$.
2	Variable alphanumérique.
6	Longueur variable 6.
26-2	26+256*2=538, adresse de stockage.

2738 à 2746

0-0	Première variable commençant par X.
216	Code ASCII de X : variable X.
4	Variable en virgule flottante.
0-0-188-43-142	Valeur de la variable 10991 : (1+43/128+188/32768)*2^(142−129).

2747 à 2755

0-0	Première variable commençant par Y.
217	Code ASCII de Y : variable Y.
4	Variable en virgule flottante.
0-0-0-34-136	Valeur de la variable 162 : (1+34/128)*2^(136−129).

2756 à 2763

120-0	Deuxième variable commençant par L.
76-204	Code ASCII de L et de L : variable LL$.
2	Variable alphanumérique.
13	Longueur variable 13.
111-166	111+256*166=42607, adresse de stockage.

2764 à 2770

0-0	Première variable commençant par W.
215	Code ASCII de W : variable W$.
2	Variable alphanumérique.
3	Longueur variable 3.
93-166	93+256*166=42589, adresse de stockage.

2771 à 2794

0-0	Première variable commençant par Z.
218	Code ASCII de Z : variable Z.
4	Variable en virgule flottante.
18-0	Nombre d'octets d'occupation : 18.
1	Première dimension.
3-0	Nombre d'éléments 1re dimension : 3.
0-0-0-0-0	Valeur de la variable Z(0)=0.
0-0-0-0-129	Valeur de la variable Z(1)=1 : 2^(129−129).
0-0-0-0-130	Valeur de la variable Z(2)=2 : 2^(130−129).

2795 à 2836

1-0	Deuxième variable commençant par Z.
90-218	Code ASCII de Z et de Z : variable ZZ.
4	Variable en virgule flottante.
35-0	Nombre d'octets d'occupation : 35.
2	Nombre de dimensions : 2.
3-0	Nombre d'éléments 2e dimension : 3.
2-0	Nombre d'éléments 1re dimension : 2.
0-0-0-0-0	Valeur de la variable ZZ(0,0)=0 : 2^−129.
0-0-0-32-132	Valeur de la variable ZZ(1,0)=10 : (1+32/64)*2^(132−129).
0-0-0-0-129	Valeur de la variable ZZ(0,1)=1 : 2^(129−129).
0-0-0-48-132	Valeur de la variable ZZ(1,1)=11 : (1+48/128)*2^(132−129).
0-0-0-0-130	Valeur de la variable ZZ(0,2)=2 : 2^(130−129).
0-0-0-64-132	Valeur de la variable ZZ(1,2)=12 : (1+64/128)*2^(132−129).

2837 à 2849

0-0	Première variable commençant par X.
216	Code ASCII de X : variable X%.
1	Variable entière.
7-0	Nombre d'octets d'occupation : 7.
1	Nombre de dimensions : 1.
2-0	Nombre d'éléments 1re dimension : 2.
0-0	Valeur de la variable X%(0) = 0.
100-0	Valeur de la variable X%(1) = 100.

2850 à 2870

0-0	Première variable commençant par Y.
217	Code ASCII de Y : variable Y$.
2	Variable alphanumérique.
15-0	Nombre d'octets d'occupation : 15.
1	Nombre de dimensions : 1.
4-0	Nombre d'éléments 1re dimension : 4.
1	Longueur Y$(0) = 1.
92-166	92 + 256*166 = 42588. Adresse stockage Y$(0).
1	Longueur Y$(1) = 1.
91-166	91 + 256*166 = 42587. Adresse stockage Y$(1).
1	Longueur Y$(2) = 1.
90-166	90 + 256*166 = 42586. Adresse stockage Y$(2).
1	Longueur Y$(3) = 1.
89-166	89 + 256*166 = 42585. Adresse stockage Y$(3).

DE HIMEM A FRETOP

42619 à 42607

89-84-82-69-87-81-82-85-79-74-78-79-66.

Code ASCII de Y, T, R, E, W, Q, R, U, O, J, N, O, B, soit BONJOURQWERTY ; c'est la variable LL$.

42608 à 42597

52-51-50-49-89-84-82-69-87-81.

Code ASCII de 4, 3, 2, 1, Y, T, R, E, W, Q, soit QWERTY1234 ; c'est la variable L$.

42596 à 42594

65-65-65.

Code ASCII de A, A, A, soit AAA ; c'est la première valeur de la variable W$.

42593 à 42589

66-66-66-66-66.

Code ASCII de B, B, B, B, B, soit BBBBB ; c'est la valeur actuelle de la variable W$.

42588 à 42585

65-66-67-68.

Code ASCII de A, B, C, D ; ce sont les quatre variables Y$(3), Y$(2), Y$(1), Y$(0).

42584 à 42571

90-90-90-90-90-90-90-90-90-90-90-90-90-90.

Quatorze fois le code ASCII Z ; ce sont les valeurs successives de la variable R$.

Au-delà de 42571, les informations ne sont plus liées au programme en cours d'activité.

La variable alphanumérique MM$ a une adresse de stockage de 538, c'est-à-dire une adresse à l'intérieur du programme BASIC. En effet, l'interpréteur ne recopie pas la valeur de cette variable dans la zone HIMEM/FRETOP, puisqu'elle est déjà disponible dans les mémoires. Par contre la variable L$, au moment de l'exécution de l'ordre 20, a son adresse dans le programme BASIC, mais à l'exécution de l'ordre 40 la variable est stockée dans la zone HIMEM/FRETOP, puisque sa nouvelle valeur n'est plus disponible dans les mémoires du programme.

Cette exploration nous montre aussi que l'interpréteur BASIC n'a pas effectué le nettoyage de la zone HIMEM/FRETOP ; il stocke les variables alphanumériques au fur et à mesure de leur arrivée. Lorsqu'une telle variable change de valeur, il stocke la nouvelle valeur à la suite des autres variables ; en effet, si la nouvelle chaîne de caractères a une longueur différente de la précédente, soit il y aurait de la place perdue, soit il n'y aurait pas assez de place s'il essayait de la remettre au même endroit. Nous avons alors l'ancienne valeur et la nouvelle dans la zone HIMEM/FRETOP, mais seule la nouvelle valeur est référencée dans la zone LOMEM/ARYTAB ; c'est le cas des variables W$ et R$. Au bout d'un certain temps, si la valeur de FRETOP rejoint la valeur de STREND l'interpréteur enlève les valeurs inutiles et remet les unes à la suite des autres les valeurs actuelles. On peut forcer l'interpréteur à effectuer ce travail en insérant l'odre FREE(0).

4

INTRODUCTION DE DEUX PROGRAMMES INDÉPENDANTS

Nous venons de réaliser des programmes qui s'auto-exploraient ; nous allons maintenant examiner l'exploration d'un programme quelconque.

La première possibilité consiste à réserver un espace suffisant sous le TXTTAB auquel travaillera le programme à explorer, ou au-dessus du HIMEM ; puis à introduire le programme d'exploration à cet endroit ; rétablir le TXTTAB ou le HIMEM ; introduire le programme à explorer ; remettre TXTTAB et HIMEM à la valeur du programme d'exploration et exécuter le programme d'exploration.

La seconde possibilité consiste à introduire tout d'abord le programme à explorer dans les mémoires, puis à introduire le programme d'exploration à un autre endroit des mémoires, enfin à exécuter le programme d'exploration.

Nous allons commencer par examiner cette dernière possibilité, qui est la plus rapide.

Nous avons vu, au Chapitre 2, que la zone mémoire située entre STREND et FRETOP était un espace libre ; nous allons donc implanter le programme d'exploration à cet endroit.

Tout d'abord, après avoir fait un RESET par CTRL + SHIFT + ESC, nous allons regarder les valeurs de TXTTAB, LOMEM, ARYTAB, STREND, FRETOP, HIMEM lorsqu'il n'y a pas de programme dans les mémoires de l'Amstrad (pour TXTTAB, LOMEM et HIMEM, nous examinons les deux séries de mémoires qui contiennent ces valeurs).

Tapons :

```
PRINT PEEK(44673)+256*PEEK(44674)      Réponse :    367 (TXTTAB)
PRINT PEEK(44592)+256*PEEK(44593)      Réponse :    367 (TXTTAB)
PRINT PEEK(44675)+256*PEEK(44676)      Réponse :    370 (LOMEM)
PRINT PEEK(44677)+256*PEEK(44678)      Réponse :    370 (LOMEM)
PRINT PEEK(44679)+256*PEEK(44680)      Réponse :    370 (ARYTAB)
PRINT PEEK(44681)+256*PEEK(44682)      Réponse :    370 (STREND)
PRINT PEEK(45197)+256*PEEK(45198)      Réponse :  42619 (FRETOP)
PRINT PEEK(45199)+256*PEEK(45200)      Réponse :  42619 (HIMEM)
PRINT PEEK(45667)+256*PEEK(45668)      Réponse :  42619 (HIMEM)
```

REMARQUE

LOMEM est situé trois mémoires au-dessus de TXTTAB ; LOMEM, ARYTAB et STREND ont la même valeur puisqu'il n'y a pas encore de variables.

Introduisons alors le programme suivant :

```
10 CLS
20 DATA BONJOUR,PROGRAMME1
30 DIM A$(2),A(10)
40 FOR I=1 TO 2 : READ A$(I) : NEXT I
50 FOR I=1 TO 10 : A(I)=I : NEXT I
60 B=1 : C$=A$(1)+A$(2)
70 PRINT A$(1),A$(2)
80 END
```

Au passage, sauvegardons ce programme sous le nom de CH4-P1.
Examinons de nouveau TXTTAB, LOMEM, ARYTAB, STREND, FRETOP, HIMEM.

```
PRINT PEEK(44673)+256*PEEK(44674)      Réponse :  367 (TXTTAB)
PRINT PEEK(44592)+256*PEEK(44593)      Réponse :  367 (TXTTAB)
PRINT PEEK(44675)+256*PEEK(44676)      Réponse :  549 (LOMEM)
PRINT PEEK(44677)+256*PEEK(44678)      Réponse :  549 (LOMEM)
PRINT PEEK(44679)+256*PEEK(44680)      Réponse :  549 (ARYTAB)
PRINT PEEK(44681)+256*PEEK(44682)      Réponse :  549 (STREND)
PRINT PEEK(45197)+256*PEEK(45198)      Réponse :  42619 (FRETOP)
PRINT PEEK(45199)+256*PEEK(45200)      Réponse :  42619 (HIMEM)
PRINT PEEK(44667)+256*PEEK(44668)      Réponse :  42619 (HIMEM)
```

REMARQUE

Vous devez impérativement respecter exactement les programmes, y compris les espaces, pour obtenir les mêmes valeurs.

Les valeurs de LOMEM, ARYTAB et STREND n'ont pas changé puisque le programme n'a pas été exécuté.
Nous tapons RUN pour exécuter ce programme. Nous voyons s'afficher :

```
BONJOUR          PROGRAMME1
```

Examinons de nouveau TXTTAB, LOMEM, ARYTAB, STREND, FRETOP, HIMEM

```
PRINT PEEK(44673)+256*PEEK(44674)      Réponse :  367 (TXTTAB)
PRINT PEEK(44592)+256*PEEK(44593)      Réponse :  367 (TXTTAB)
PRINT PEEK(44675)+256*PEEK(44676)      Réponse :  572 (LOMEM)
PRINT PEEK(44677)+256*PEEK(44678)      Réponse :  572 (LOMEM)
PRINT PEEK(44679)+256*PEEK(44680)      Réponse :  597 (ARYTAB)
PRINT PEEK(44681)+256*PEEK(44682)      Réponse :  679 (STREND)
PRINT PEEK(45197)+256*PEEK(45198)      Réponse :  42602 (FRETOP)
PRINT PEEK(45199)+256*PEEK(45200)      Réponse :  42619 (HIMEM)
PRINT PEEK(44667)+256*PEEK(44668)      Réponse :  42619 (HIMEM)
```

Nous constatons que la zone qui va de l'adresse 679 à l'adresse 42602 est libre. Nous allons changer les valeurs de TXTTAB, LOMEM, ARYTAB, STREND, FRETOP, HIMEM pour introduire un deuxième programme sans détruire le premier ; mais pour pouvoir le faire, il est nécessaire qu'il n'y ait pas eu d'ordre RUN. On effectue donc un RESET avec CTRL+SHIFT+ESC, puis LOAD "CH4-P1". Tapons alors :

```
POKE44673,0 : POKE 44674,10     Positionne TXTTAB  en  2560
POKE44675,3 : POKE 44676,10     Positionne LOMEM  en  2563
POKE44679,3 : POKE 44680,10     Positionne ARYTAB en  2563
POKE44681,3 : POKE 44682,10     Positionne STREND en  2563
POKE45197,0 : POKE 45198,150    Positionne FRETOP en  38400
POKE45199,0 : POKE 45200,150    Positionne HIMEM  en  38400
```

REMARQUE

Il n'est pas nécessaire, pour l'introduction d'un programme, de changer les valeurs des deuxièmes couples de mémoires de TXTTAB, LOMEM et HIMEM.

La zone réservée aux programmes BASIC est maintenant celle qui va de l'adresse 2560 à l'adresse 38400. Si nous introduisons un nouveau programme, il va se positionner dans cet espace mémoire, ce que nous allons vérifier. Tapons :

```
10 CLS
20 DATA PROGRAMME2,AUREVOIR
30 DIM A$(2),A(10)
40 FOR I=1 TO 2 : READ A$(I) : NEXT I
50 FOR I=1 TO 10 : A(I)=I : NEXT I
60 B=2 : C$=A$(1)+A$(2)
70 PRINT A$(1),A$(2)
80 END
```

Sauvegardons ce programme sous le nom de CH4-P2.

Nous tapons RUN pour exécuter ce programme et nous voyons s'afficher :

```
PROGRAMME2        AUREVOIR
```

C'est bien le second programme qui s'exécute.

Examinons TXTTAB, LOMEM, ARYTAB, STREND, FRETOP, HIMEM :

```
PRINT PEEK(44673)+256*PEEK(44674)      Réponse :  2560 (TXTTAB)
PRINT PEEK(44592)+256*PEEK(44593)      Réponse :  2560 (TXTTAB)
PRINT PEEK(44675)+256*PEEK(44676)      Réponse :  2766 (LOMEM)
PRINT PEEK(44677)+256*PEEK(44678)      Réponse :  2766 (LOMEM)
PRINT PEEK(44679)+256*PEEK(44680)      Réponse :  2791 (ARYTAB)
PRINT PEEK(44681)+256*PEEK(44682)      Réponse :  2873 (STREND)
PRINT PEEK(45197)+256*PEEK(45198)      Réponse :  38382 (FRETOP)
PRINT PEEK(45199)+256*PEEK(45200)      Réponse :  38400 (HIMEM)
PRINT PEEK(44667)+256*PEEK(44668)      Réponse :  38400 (HIMEM)
```

Nous constatons que les deuxièmes couples de mémoires pour TXTTAB, LOMEM et HIMEM ont bien pris les bonnes valeurs.

Si maintenant nous remettons TXTTAB, LOMEM, ARYTAB, STREND, FRETOP, HIMEM aux valeurs du premier programme, et cela aussi pour les deuxièmes couples de TXTTAB, LOMEM et HIMEM :

```
POKE44673,111 : POKE 44674,1       TXTTAB  (367=111+256*1)
POKE44592,111 : POKE 44593,1       TXTTAB  (367=111+256*1)
POKE44675,60 : POKE 44676,2        LOMEM   (549=37+256*2)
POKE44677,60 : POKE 44678,2        LOMEM   (549=37+256*2)
POKE44679,60 : POKE 44680,2        ARYTAB  (31622=55+256*2)
POKE44681,60 : POKE 44682,2        STREND  (31691=137+256*2)
POKE30980,123 : POKE 45198,166     FRETOP  (42619=123+256*166)
POKE45199,123 : POKE 45200,166     HIMEM   (42619=123+256*166)
```

et si nous faisons :

```
LIST
```

nous obtenons le listing du Programme 1.

Nous pouvons maintenant faire RUN... et le Programme 1 s'exécute.

Si nous voulons revenir au Programme 2, il suffit de repositionner TXTTAB, LOMEM, ARYTAB, STREND, FRETOP et HIMEM.

```
POKE44673,0 : POKE 44674,10       TXTTAB   (2560=0+256*10)
POKE44592,0 : POKE 44593,10       TXTTAB   (2560=111+256*10)
POKE44675,206 : POKE 44676,10     LOMEM    (2766=206+256*10)
POKE44677,206 : POKE 44678,10     LOMEM    (2766=206+256*10)
POKE44679,231 : POKE 44680,10     ARYTAB   (2791=231+256*10)
POKE44681,57 : POKE 44682,11      STREND   (2873=57+256*11)
POKE30980,238 : POKE 45198,149    FRETOP   (38382=238+256*149)
POKE45199,0 : POKE 45200,150      HIMEM    (38400=0+256*150)
```

Si nous tapons :

```
PRINT C$,B
```

nous obtenons la réponse :

```
"PROGRAMME2AUREVOIR      2"
```

Les variables de travail du second programme sont donc toujours bien présentes.

*

*   *

Nous avons donc le schéma suivant d'occupation de la mémoire : d'une part avec uniquement le programme CH4-P1, d'autre part avec le programme CH4-P1 + CH4-P2.

Il s'agit bien sûr d'un cas particulier au niveau des valeurs des adresses.

COEXISTENCE DE DEUX PROGRAMMES EN MÉMOIRE
SCHÉMA D'OCCUPATION DES MÉMOIRES

	CH4-P1		**CH4-P1 + CH4-P2**	
HIMEM		42619		
	Variables alphanumériques CH4-P1		Variables alphanumériques CH4-P1	
FRETOP		42602		
	libre		libre	
		38400		HIMEM
			Variables alphanumériques CH4-P2	
		38382		FRETOP
			libre	
		2873		STREND
			Variables dim CH4-P2	
		2791		ARYTAB
			Variables CH4-P2	
		2766		LOMEM
			CH4-P2 BASIC	
		2560		TXTTAB
			libre	
STREND		679		
	Variables dim CH4-P1		Variables dim CH4-P1	
ARYTAB		597		
	Variables CH4-P1		Variables CH4-P1	
LOMEM		572		
	CH4-P1 BASIC		CH4-P1 BASIC	
TXTTAB		367		

En conclusion de ce chapitre, nous venons de montrer que plusieurs programmes peuvent coexister dans les mémoires de l'Amstrad et que l'on peut passer de l'un à l'autre. Pour cela, il est nécessaire de positionner pour une première introduction :

TXTTAB	(dont l'adresse est contenue dans les mémoires 44673 et 44674) à la valeur de la mémoire où l'on désire que la première instruction BASIC soit écrite.
LOMEM	(dont l'adresse est contenue dans les mémoires 44675 et 44676) à la valeur TXTTAB+3.
ARYTAB	(dont l'adresse est contenue dans les mémoires 44679 et 44680) à la valeur TXTTAB+3.
STREND	(dont l'adresse est contenue dans les mémoires 44681 et 44682) à la valeur TXTTAB+3.
HIMEM	(dont l'adresse est contenue dans les mémoires 45199 et 45200) à la valeur désirée, valeur maximale d'utilisation des mémoires par ce programme.
FRETOP	(dont l'adresse est contenue dans les mémoires 45197 et 45198) à la valeur désirée, valeur maximale d'utilisation des mémoires par ce programme.

Deux programmes ou plus peuvent être introduits, *à condition de ne pas avoir effectué de RUN.*

Si l'on revient à un programme qui a déjà été exécuté, il est de plus nécessaire de remettre TXTTAB, LOMEM, ARYTAB, STREND, FRETOP et HIMEM ainsi que les deuxièmes couples de mémoires pour TXTTAB, LOMEM et HIMEM (soit 44592/44593 pour TXTTAB, 44677/44678 pour LOMEM, 44667/44668 pour HIMEM) à la valeur qu'ils avaient à la fin de l'exécution du programme ; valeur à rechercher immédiatement avant toute autre manipulation (LOMEM : PEEK(44675) et PEEK(44676), ARYTAB : PEEK(44679) et PEEK(44680), STREND : PEEK(44681) et PEEK(44682), FRETOP : PEEK(45197) et PEEK(45198)).

Tout ce qui vient d'être expliqué est valable pour la première possibilité que nous avons citée au début de ce chapitre, c'est-à-dire de placer un programme sous le TXTTAB ou au-dessus du HIMEM ; seul le schéma d'occupation des mémoires sera modifié.

Ces différentes techniques seront utilisées dans la suite de cet ouvrage, en particulier chaque fois qu'il sera nécessaire de protéger des zones mémoire contre l'effacement.

5

PROGRAMME DE RECHERCHE D'ORDRES BASIC

Nous allons maintenant réaliser la première partie de notre programme de déverminage, qui est la recherche des ordres BASIC. Pour cela, nous allons explorer les mémoires situées entre TXTAB et LOMEM (comme au Chapitre 3) en recherchant un ordre BASIC particulier. Comme nous l'avons vu, les ordres BASIC sont représentés par des tokens, c'est-à-dire des codes ASCII supérieurs à 127.

Notre programme, que nous appellerons TOKEN, va comporter les étapes suivantes :

- Un menu permettant de choisir l'ordre BASIC à trouver.

- Une exploration des mémoires entre TXTAB et LOMEM ; lorsqu'on trouvera un code ASCII égal à celui du token correspondant à l'ordre recherché, on stockera l'étiquette de l'ordre. Mais pour cela, nous devons être sûrs qu'il s'agit du code ASCII d'un token ; par conséquent, nous devons éliminer les octets correspondant aux adresses, aux constantes et aux fins de variables.

- Lorsque tout le programme aura été exploré, nous éditerons le résultat soit à l'écran, soit sur une imprimante.

Pour des raisons d'encombrement mémoire, afin de ne pas avoir une taille prohibitive pour ce programme, nous avons limité le menu à neuf ordres BASIC et le nombre d'étiquettes à 100.

D'autre part, ce programme est la première partie de notre programme de déverminage ; nous allons lui donner des étiquettes comprises entre 10000 et 20000, ce qui laissera la place aux étiquettes des instructions des autres programmes.

Pour pouvoir explorer un programme existant, déjà chargé en mémoire aux valeurs normales de TXTAB et de HIMEM, nous implanterons notre programme d'exploration à un emplacement non occupé par le programme à explorer. Nous le mettrons dans la zone au-dessus de l'adresse 20480 (ce qui laisse environ 20 000 mémoires pour le programme testé et ses variables numériques). Mais cette opération ne sera à faire qu'au moment de l'utilisation, nous la verrons donc au Chapitre 7.

Réalisons maintenant les trois parties du programme TOKEN. Le LOMEM de ce programme est 6319.

MENU

Le menu sera implanté entre les étiquettes 9994 et 11000 ; en voici le listing, que nous allons expliquer.

```
9994 REM                           PROGRAMME DE RECHERCHE DE TOKEN
9996 REM                                TXTTAB A 111+  1*256=  367
9998 REM                                 LOMEM A 175+ 24*256= 6319
10000 TXTTAB=368
10010 DIM A(100),A$(10)
10020 DATA CLS,DATA,DIM,ELSE,FOR,GOTO,LOCATE,PRINT,READ
10030 FOR I=1 TO 9 : READ A$(I) : NEXT I
10040 REM ..... CHOIX DE L'ORDRE RECHERCHE .....
10050 FOR I=1 TO 100 : A(I)=0 : NEXT I : CLS
10060 LOCATE 7,1 : PRINT "CHOIX DU TOKEN CHERCHE :"
10070 LOCATE 10,4 : PRINT "CLS           1"
10080 LOCATE 10,6 : PRINT "DATA          2"
10090 LOCATE 10,8 : PRINT "DIM           3"
10100 LOCATE 10,10 : PRINT "ELSE          4"
10110 LOCATE 10,12 : PRINT "FOR           5"
10120 LOCATE 10,14 : PRINT "GOTO          6"
10130 LOCATE 10,16 : PRINT "LOCATE        7"
10140 LOCATE 10,18 : PRINT "PRINT         8"
10150 LOCATE 10,20 : PRINT "READ          9"
10160 LOCATE 1,23 : PRINT "TAPER LE NUMERO DU TOKEN CHERCHE [1/9]"
10170 R$=INKEY$
10180 R$=INKEY$ : IF R$="" THEN 10180 ELSE R=ASC(R$)
10190 IF R<49 OR R>57 THEN 10180 ELSE LOCATE 40,23 : PRINT R$
10200 TK=0
10210 TK=-138*(R=49)-140*(R=50)-147*(R=51)-151*(R=52)-158*(R=53)
10220 TK=TK-160*(R=54)-169*(R=55)-191*(R=56)-195*(R=57)
10230 RR=R-48
```

9994 à 9999

Quelques REM pour rappeler le nom du programme et ses adresses d'implantation.

10000

Nous mettons la valeur du TXTTAB + 1 du programme à explorer dans la variable de nom TXTTAB, soit 368 dans le cas normal.

Si vous voulez explorer des programmes implantés à un autre TXTTAB, il suffit de changer cette valeur.

10010

La variable dimensionnée A sera utilisée pour stocker les étiquettes des instructions où se trouve l'ordre BASIC recherché. Nous l'avons dimensionnée pour 100 étiquettes, c'est habituellement suffisant pour notre utilisation.

La variable A$ est destinée à recevoir le nom de l'ordre que l'on recherchera. C'est elle qui nous permettra d'indiquer en tête du résultat final le nom de l'ordre que l'on a cherché.

10020

Ces DATA servent à remplir la variable A$.

10030

Boucle FOR/NEXT en I pour introduire les DATA dans la variable A$.

10040

Un REM pour rappeler le but du morceau de programme qui va suivre.

10050

Initialise à 0 les 100 variables A(I) de manière à simplifier ultérieurement l'affichage des résultats successifs.

10060 à 10160

Les ordres LOCATE et PRINT servent à réaliser l'affichage à l'écran du menu et de ses explications.

10170

Cette instruction sert à éliminer la lecture de la dernière touche enfoncée.

10180 à 10190

Ces instructions permettent la lecture du clavier en mettant le résultat dans la variable R par R=ASC(R$) ; la fonction ASC ne pouvant pas travailler sur R$ égal rien du tout, on doit éliminer le cas R$="".

Puis on teste la validité de l'information transmise par IF R<49 OR R>57 THEN 10180. En effet le 1 a comme code ASCII : 49, le 2 : 50, et ainsi de suite jusqu'à 9 code 57 (voir Annexe A) ; la condition logique sera donc remplie pour les réponses 1, 2,... 9 et le programme continuera alors à l'instruction 10200 ; elle ne sera pas remplie pour toute autre touche ; dans ce cas, il y aura retour au début de l'instruction 10180.

Pour finir, on affiche le caractère correspondant à la touche enfoncée, si la réponse est acceptée.

10200 à 10220

Ces instructions calculent la valeur du token, variable TK, correspondant au choix effectué à l'aide d'une équation logique.

En effet, l'expression (R=49) vaut 0 si R est différent de 49 et elle vaut −1 si R=49 (nous avons bien dit : *moins un*). Donc pour R=49 l'expression −138*(R=49) vaut (−138*−1)= +138 (et zéro pour les autres valeurs de R), ce qui est bien la valeur du token de FOR qui correspond au choix 1, soit R=49. Le même raisonnement montre que, dans le cas où R=49, tous les autres termes de l'équation sont nuls, et l'on a bien la valeur 138 pour la variable TK.

Pour R=50 choix 2, c'est le deuxième terme qui n'est pas nul ; il prend la valeur (−140*−1)=140, valeur du token de NEXT, et ainsi de suite pour les autres choix.

10230

Compte tenu de la remarque faite pour l'instruction 10180, la variable RR=R−48 prend la valeur du choix effectué (c'est-à-dire 1, 2, 3... ou 9), ce qui nous permettra de retrouver la variable indicée A$(R) qui correspond bien au choix effectué. En effet, si l'on a enfoncé la touche correspondant à 2, R aura la valeur du code ASCII de la touche 2, soit 50 (voir Annexe A) et RR=R−48=50−48=2.

EXPLORATION DES MÉMOIRES

Nous allons implanter l'exploration des mémoires entre les étiquettes 10999 et 12000 ; en voici le listing :

```
10999 REM ...........................................................
11000 REM                                                 EXPLORATION
11001 REM ...........................................................
11010 CLS
11020 LOCATE 8,10:PRINT "ETIQUETTE DE L'INSTRUCTION"
11030 LOCATE 10,12:PRINT "EN COURS D'EXPLORATION"
11040 T=TXTTAB : I=0 : AD=TXTTAB
11044 REM ...........................................................
11045 REM                             ADRESSE DE L'INSTRUCTION SUIVANTE
11046 REM ...........................................................
11050 AD=PEEK(T)+256*PEEK(T+1)+AD
11054 REM ...........................................................
11055 REM                                    ETIQUETTE DE L'INSTRUCTION
11056 REM ...........................................................
11060 ET=PEEK(T+2)+256*PEEK(T+3)
11070 LOCATE 18,16 : PRINT ET
11080 T=T+3
11090 T=T+1 : X=PEEK(T)
```

```
11094 REM ..........................................................
11095 REM                                           ELIMINATION DES REM
11096 REM ..........................................................
11100 IF X=197 THEN T=AD : GOTO 11050
11104 REM ..........................................................
11105 REM                                 ELIMINATION DE L'ALPHANUMERIQUE
11106 REM ..........................................................
11110 IF X=34 THEN GOTO 11120 ELSE GOTO 11130
11120 T=T+1  : IF PEEK(T)<>34 THEN GOTO 11120
11124 REM ..........................................................
11125 REM                                     ELIMINATION DES VARIABLES
11126 REM ..........................................................
11130 IF X<>13 AND X<>2 AND X<>3 THEN GOTO 11150 ELSE T=T+2
11140 T=T+1  : IF PEEK(T)<129 THEN 11140 ELSE T=T+1 : X=PEEK(T)
11144 REM ..........................................................
11145 REM                                    ELIMINATION DES CONSTANTES
11146 REM ..........................................................
11150 IF X=25 THEN T=T+1  : GOTO 11090
11160 IF X>25 AND X<31 THEN T=T+2 : GOTO 11090
11170 IF X=31 THEN T=T+5 : GOTO 11090
11174 REM ..........................................................
11175 REM                                      TEST DE FIN DE PROGRAMME
11176 REM ..........................................................
11180 D=PEEK(T): E=PEEK(T+1)  : F= PEEK(T+2)
11190 IF D=0 AND E=0 AND F=0 THEN GOTO 12010
11194 REM ..........................................................
11195 REM                                     TEST DE FIN D'INSTRUCTION
11196 REM ..........................................................
11200 IF X=0 THEN T=T+1  : GOTO 11050
11204 REM ..........................................................
11205 REM                                     TOKEN/STOCKAGE ETIQUETTE
11206 REM ..........................................................
11210 IF X=TK THEN I=I+1  : A(I)=ET
11220 GOTO 11090
```

10999 à 11001

Des REM pour rappeler le but du module de programme qui va suivre.

11010

CLS nettoyage de l'écran.

11020 et 11030

L'exploration durant un certain temps, il est nécessaire de montrer à l'opérateur que le programme fonctionne correctement ; pour cela, on lui présente à l'écran, à chaque instant, l'étiquette en cours d'exploration. Ces deux ordres inscrivent à l'écran la signification des chiffres qui vont apparaître.

11040

T va être la variable correspondant à l'adresse de la mémoire en cours d'exploration ; on l'initialise donc à la valeur de la première adresse du programme à explorer, soit la valeur normale du TXTTAB + 1.

I va être la variable qui va compter le nombre d'étiquettes où se trouve l'ordre BASIC recherché ; on l'initialise à 0.

AD sera la variable contenant l'adresse du début de l'instruction suivante, on l'initialise donc à TXTTAB+1.

11044 et 11050

Nous avons vu (voir Chapitre 3) qu'une instruction commençait par la longueur de l'instruction dans les mémoires ; nous calculons l'adresse de l'instruction suivante, cette adresse nous servira ultérieurement.

11054 et 11060

Les deux mémoires suivantes correspondent au numéro d'étiquette ; nous calculons donc la valeur de l'étiquette et nous la stockons dans la variable ET.

11070

Nous l'inscrivons alors à l'écran pour informer l'opérateur de l'évolution du programme.

11080 et 11090

Nous incrémentons la variable T (adresse de la mémoire à explorer) de trois unités, puis d'une unité, puisque nous venons d'exploiter les quatre premières mémoires ; cette incrémentation est faite en deux fois, de manière à pouvoir réutiliser l'incrémentation d'une unité au cours de la boucle d'exploration d'une instruction.

Ensuite nous lisons la valeur contenue par la mémoire d'adresse T, valeur que nous mettons dans la variable X (X=PEEK(T)).

11094 et 11100

A partir du moment où nous rencontrons un REM, soit le token de valeur 197, le reste de l'instruction ne peut pas contenir d'ordre BASIC ; nous effectuons dans ce cas un saut à l'étiquette n° 11050 après avoir donné à notre variable T la valeur de l'adresse de début de l'instruction suivante pour continuer l'exploration.

11104 et 11120

Lorsque nous rencontrons le code ASCII 34, soit '', nous savons que les codes ASCII qui vont suivre correspondront à du texte en alphanumérique. Nous pouvons donc passer rapidement sur les mémoires suivantes jusqu'à ce que nous rencontrions de nouveau le code ASCII 34 qui signale la fin du texte ; ces ordres permettent d'accélérer le programme mais ne sont pas indispensables à notre exploration actuelle (ils le seront par contre en recherche des variables).

11124 à 11140

Nous avons vu au Chapitre 3 que le début d'une variable était signalé par un octet égal à 13, 2 ou 3 ; si nous rencontrons le début d'une variable, nous éliminons les deux octets qui correspondent à son adresse de stockage (qui peuvent avoir des valeurs supérieures à 128), puis tous les octets suivants jusqu'à ce que l'on rencontre l'octet supérieur à 128 correspondant au dernier caractère de la variable. Le but de cette opération est d'éliminer tous les octets qui peuvent être supérieurs à 128 et qui ne sont pas des tokens. Nous continuons ce travail avec les constantes.

11144 à 11170

Nous avons vu au Chapitre 3 que le début d'une constante est signalé par :

- 25 pour une variable stockée sur un octet ; cet octet peut être supérieur à 128, nous le sautons donc.
- 26, 27, 28, 29 ou 30 pour une variable stockée sur deux octets ; ces octets peuvent être supérieurs à 128, nous les sautons.
- 31 pour une variable stockée sur cinq octets qui peuvent être supérieurs à 128, nous les sautons.
- Les autres valeurs possibles pour les constantes sont inférieures à 128 ; elles ne gêneront pas le déroulement du programme.

11174 à 11190

Nous avons vu au Chapitre 3 que la fin d'un programme BASIC se reconnaissait au fait que quatre mémoires successives contiennent

la valeur 0 ; compte tenu de nos autres tests, il suffit d'en avoir trois successives nulles, c'est ce que nous testons ici.

Si c'est la fin du programme, nous allons en 12010 où se trouve le programme d'édition du résultat de notre recherche.

11194 et 11200

Nous testons de même si nous sommes à la fin d'une instruction (identifiée par un 0).

Dans le cas de fin d'instruction, nous allons en 11050 pour l'exploration de l'instruction suivante.

11204 et 11210

Nous finissons par le test sur la valeur du token recherché ; si nous en avons trouvé un, nous stockons la valeur de l'étiquette dans A(I), puis nous incrémentons I d'une unité.

11220

Et nous retournons à l'instruction 11090 qui va incrémenter T d'une unité et continuer l'exploration de l'instruction.

PRÉSENTATION DES RÉSULTATS

Nous allons implanter la présentation des résultats entre les étiquettes 11999 et 20000 ; en voici le listing :

```
11999 REM ...........................................................
12000 REM                                        EDITION DES RESULTATS
12001 REM ...........................................................
12010 CLS
12020 LOCATE 8,4 : PRINT "EDITION DES RESULTATS"
12030 LOCATE 5,10 : PRINT "A L'ECRAN ................. E"
12040 LOCATE 5,14 : PRINT "A L'IMPRIMANTE + ECRAN .... I"
12050 LOCATE 5,20 : PRINT "TAPER VOTRE CHOIX [E/I]"
12060 R$=INKEY$
12070 R$=INKEY$ : IF R$="" THEN 12070 ELSE R=ASC(R$)
12080 IF R=69 THEN 13010
12090 IF R=73 THEN 14010
12100 GOTO 12070
12999 REM ...........................................................
13000 REM                                                 SORTIE ECRAN
13001 REM ...........................................................
13010 CLS
13020 LOCATE 3,1 : PRINT "ETIQUETTE DES INSTRUCTIONS CONTENANT"
13030 LOCATE 17,3 : PRINT A$(RR)
13040 IF I=0 THEN LOCATE 17,10 : PRINT "AUCUN" : GOTO 13080
```

```
13050 FOR J=1 TO I STEP 4
13060 PRINT USING "#######";A(J);A(J+1);A(J+2);A(J+3)
13070 NEXT J
13080 R$=INKEY$
13090 R$=INKEY$ : IF R$="" THEN 13090
13100 GOTO 15010
13999 REM ...........................................................
14000 REM                                    SORTIE ECRAN + IMPRIMANTE
14001 REM ...........................................................
14010 CLS
14020 LOCATE 8,8 : PRINT "ALLUMER L'IMPRIMANTE"
14030 LOCATE 7,12 : PRINT "PUIS ENFONCER <RETURN>"
14040 LOCATE 17,16 : INPUT Z$
14050 PRINT #8,"ETIQUETTE DES INSTRUCTIONS CONTENANT"
14060 PRINT #8,:PRINT #8,A$(RR):PRINT #8,:PRINT #8,
14070 IF I=0 THEN PRINT #8,"       --- AUCUN ---" : GOTO 13010
14080 FOR J=1 TO I STEP 4
14090 PRINT #8,A(J),A(J+1),A(J+2),A(J+3)
14100 NEXT J
14110 GOTO 13010
14999 REM ...........................................................
15000 REM                                      QUITTER OU RECOMMENCER
15001 REM ...........................................................
15010 CLS
15020 LOCATE 8,6 : PRINT "DESIREZ-VOUS EXPLORER"
15030 LOCATE 11,10 : PRINT "UN AUTRE TOKEN"
15040 LOCATE 12,15 : PRINT "[OUI/NON] :"
15050 R$=INKEY$
15060 R$=INKEY$ : IF R$="" THEN 15060 ELSE R=ASC(R$)
15070 IF R=79 THEN RUN
15080 IF R=78 THEN END
15090 GOTO 15040
15099 REM ...........................................................
```

12000

Un REM pour rappeler le but du morceau de programme qui va suivre.

12010

Nettoyage de l'écran.

12020 à 12050

Réalisation du menu permettant de choisir le type d'édition : écran ou imprimante + écran.

12060 à 12070

Interrogation du clavier comme pour le module précédent.

12080

Si la réponse est E code ASCII 69, on se branche en 13010 pour l'édition à l'écran.

12090

Si la réponse est I code ASCII 73, on se branche en 14010 pour l'édition sur l'imprimante.

12100

Si l'on arrive à cette instruction, c'est qu'on n'a obtenu aucune des réponses attendues ; on retourne donc à l'interrogation du clavier.

13000

Un REM pour rappeler le but du programme qui va suivre.

13010

Nettoyage de l'écran.

13020

Affichage du titre.

13030

Affichage de l'ordre qu'on vient de rechercher à l'aide de la variable A$.

13040

Si cet ordre n'existe pas dans le programme (soit I=0), on affiche le message ''AUCUN''.

13050 à 13070

Boucle en J pour afficher les étiquettes trouvées ; on en affiche quatre par ligne grâce à STEP 4 ; la valeur finale de la boucle est I, qui est le nombre d'étiquettes trouvées ; on utilise un PRINT USING pour cadrer les nombres. En utilisant cette méthode, si le nombre d'étiquettes affichées n'est pas un multiple de 4, nous trouverons des 0 sur la dernière ligne. L'élimination de ces 0 compliquerait inutilement le programme, car ils ne gênent pas l'exploitation du résultat.

13080 à 13100

Attente qu'une touche soit enfoncée pour continuer le programme en 15010. Cela permet la lecture des résultats après l'affichage.

14000

Un REM pour rappeler le but du programme qui va suivre.

14010

Nettoyage de l'écran.

14020 à 14030

Message pour rappeler qu'il ne faut pas oublier d'allumer l'imprimante.

14040

L'ordre INPUT Z$ arrête l'exécution jusqu'à ce que l'on ait enfoncé la touche RETURN.

14050 et 14060

Les ordres PRINT #8, envoient le titre sur l'imprimante, puis le nom de l'ordre qui a été recherché (variable A$).

14070

Message spécial pour le cas où I=0 ; pas d'instruction contenant l'ordre, puis branchement en 13010 pour éditer à l'écran.

14080 à 14100

Boucle en J identique à celle rencontrée aux instructions 13050/13070 ; mêmes remarques.

14110

On part en 13010 pour compléter par une édition à l'écran.

15010

Nettoyage de l'écran.

15020 à 15040

Menu pour la suite des festivités : on recommence avec O (oui), on arrête avec N (non).

15050 à 15060

Interrogation du clavier.

15070

Si la réponse est O code ASCII 79, on repart au début en 10050 pour choisir un ordre.

15080

Si la réponse est N code ASCII 78, on termine le programme.

15090

Si la réponse est différente de O ou N, on réinterroge le clavier en allant en 15030.

Nous introduisons ce programme au TXTTAB habituel, nous le mettrons à l'endroit adéquat ultérieurement.

Exécution par RUN ; auto-exploration du programme, ce qui vous permet de vérifier son bon fonctionnement.

Quant ce premier programme fonctionne, vous êtes prêts à passer au Chapitre 6. Vous devez cependant vérifier que le LOMEM est bien positionné. Si vous n'avez pas la même valeur que nous, retirez ou ajoutez des espaces dans les ordres pour obtenir un LOMEM de 6319, cela vous évitera des complications quand nous serons arrivés au Chapitre 7.

6

PROGRAMME DE RECHERCHE DE VARIABLES

Le programme, que nous appellerons VARIABLE, va comporter les étapes suivantes comme pour le programme de recherche d'ordres BASIC :

1. Un menu qui permettra de choisir le type de variables à chercher. Selon le choix effectué (réponse 1, 2, 3, 4, 5 ou 6), le programme s'orientera vers :

- le module 22000 pour traiter les variables entières ;
- le module 23000 pour traiter les variables réelles ;
- le module 24000 pour traiter les variables alphanumériques ;
- le module 25000 qui traitera les variables dimensionnées entières ;
- le module 26000 qui traitera les variables dimensionnées réelles ;
- le module 27000 qui traitera les variables dimensionnées alphanumériques.

Ces six programmes de traitement feront appel à un ensemble de programmes d'exploration, sensiblement analogues à ceux que nous venons de réaliser au Chapitre 5, et qui seront implantés sous forme de sous-programmes entre les étiquettes 21000 et 21999.

2. Un programme d'édition des résultats qui sera géré par les modules :

28000 Menu qui donne le choix entre l'écran ou l'écran et l'imprimante comme périphérique de sortie des résultats.

29000 Pour la sortie ECRAN.

30000 Pour la sortie ECRAN + IMPRIMANTE.

Nous allons maintenant donner le listing des différentes parties de ce programme et les explications correspondantes.

MENU

```
19994 REM                          PROGRAMME DE RECHERCHE DE VARIABLES
19996 REM                                      TXTTAB=111+  1*256=  367
19998 REM                                       LOMEM=140+ 36*256= 9356
20000 TXTTAB=368
20010 DIM B$(25):FOR I=1 TO 25 : B$(I)="" : NEXT I
20020 CLS
20030 LOCATE 8,1 : PRINT "CHOIX DU TYPE DE VARIABLE"
20040 LOCATE 5,6 : PRINT "ENTIERE ..................... 1"
20050 LOCATE 5,8 : PRINT "REELLE ...................... 2"
20060 LOCATE 5,10 : PRINT "ALPHANUMERIQUE .............. 3"
```

```
20070 LOCATE 5,12: PRINT "ENTIERE DIMENSIONNE ......... 4"
20080 LOCATE 5,14 : PRINT "RELLE DIMENSIONNE ........... 5"
20090 LOCATE 5,16 : PRINT "ALPHANUMERIQUE DIMENSIONNE .. 6"
20100 LOCATE 1,20 : PRINT "TAPER LE NUMERO DE VOTRE CHOIX [1/6]"
20110 R$=INKEY$
20120 R$=INKEY$ : IF R$="" THEN 20120 ELSE R=ASC(R$)
20130 IF R<49 OR R>54 THEN 20120 ELSE LOCATE 40,20 : PRINT R$
20140 T=TXTTAB : AD=TXTTAB : I=0
20150 Y=R-48 : ON Y GOTO 22010,23010,24010,25010,26010,27010
```

19994 à 19998

REM rappelant le nom et les caractéristiques du programme.

REMARQUE

Le LOMEM du présent programme n'est bien sûr connu qu'au terme de son introduction totale.

20000

Nous mettons la valeur du TXTTAB + 1 du programme à explorer dans la variable TXTTAB, soit 368 dans le cas normal.

Si vous voulez explorer des programmes implantés à un autre TXTTAB, il suffit de changer cette valeur.

20010

La variable B$(I) contiendra le nom des variables successives, d'un type donné, que nous allons trouver dans la recherche. Nous la dimensionnons à 25 ; si vos programmes ont plus de variables, vous pouvez prendre un dimensionnement plus grand. Nous commençons par une mise à ''rien du tout'' de cette variable.

20020

Nettoyage de l'écran.

20030 à 20100

Affichage du menu et du mode d'emploi à l'aide d'ordres LOCATE et PRINT.

20110 à 20130

Lecture du clavier ; R contient la valeur du code ASCII envoyé par le clavier. Nous testons ensuite la valeur de R ; R doit être compris

entre 49 (code ASCII de 1) et 54 (code ASCII de 6) pour que le choix soit compris entre 1 et 6. Si ce n'est pas le cas, nous recommençons la lecture du clavier par THEN 20120 (la procédure de lecture est la même que pour le programme TOKEN.

20140

La variable T donnera le numéro de la mémoire à explorer ; nous l'initialisons à la valeur de TXTTAB première mémoire du programme. AD sera la variable contenant l'adresse du début de l'instruction suivante ; on l'initialise donc à TXTTAB + 1. La variable I servira à comptabiliser le nombre de variables trouvées dans la recherche ; ce sera aussi l'indice du B$, nous l'initialisons à zéro.

20150

La variable Y = R – 48 prend la valeur 1 lorsque la réponse a été "1", 2 pour la réponse "2", et ainsi de suite ; elle est utilisée pour l'aiguillage qui permet d'envoyer sur le programme de recherche adéquat.

SOUS-PROGRAMMES

Premier sous-programme

Ce sous-programme sert à explorer les mémoires successives et à faire un premier tri dans les codes ASCII trouvés dans les mémoires.

```
20999 REM ...............................................................
21000 REM                                                 SOUS PROGRAMMES
21001 REM ...............................................................
21100 CLS : LOCATE 8,10 : PRINT "ETIQUETTE DE L'INSTRUCTION"
21110 LOCATE 10,12 : PRINT "EN COURS D'EXPLORATION"
21119 REM ...............................................................
21120 REM                                        TEST DE FIN DE PROGRAMME
21121 REM ...............................................................
21130 D=PEEK(T-1) : E=PEEK(T) : F=PEEK(T+1)
21135 IF D=0 AND E=0 AND F=0 THEN FL=-1 : RETURN
21139 REM ...............................................................
21140 REM                              ADRESSE DE L'INSTRUCTION SUIVANTE
21141 REM ...............................................................
21150 AD=PEEK(T)+256*PEEK(T+1)+AD
21169 REM ...............................................................
21170 REM                                      ETIQUETTE DE L'INSTRUCTION
21171 REM ...............................................................
21180 ET=PEEK(T+2)+256*PEEK(T+3)
21190 LOCATE 18,16 : PRINT ET
21200 T=T+3
21210 T=T+1 : X=PEEK(T)
21219 REM ...............................................................
21220 REM                                            ELIMINATION DES REM
21221 REM ...............................................................
```

```
21230 IF X=197 THEN T=AD : GOTO 21130
21239 REM ............................................................
21240 REM                                  ELIMINATION L'ALPHANUMERIQUE
21241 REM ............................................................
21250 IF X=34 THEN 21260 ELSE 21290
21260 T=T+1 : IF PEEK(T)<>34 THEN 21260
21279 REM ............................................................
21280 REM                                           ELIMINATION DES DATA
21281 REM ............................................................
21290 IF X=140 THEN 21300 ELSE 21340
21300 T=T+1 : X=PEEK(T)
21310 IF X=0 THEN 21340
21320 IF X=1 THEN 21210 ELSE 21300
21329 REM ............................................................
21330 REM                                     ELIMINATION DES CONSTANTES
21331 REM ............................................................
21340 T=T-(X>13 AND X<24)-2*(X=25)-3*(X>25 AND X<31)-6*(X=31):X=PEEK(T)
21349 REM ............................................................
21350 REM                                 ELIMINATION DES TOKENS DOUBLES
21351 REM ............................................................
21360 IF X=255 THEN T=T+2 : X=PEEK(T)
21369 REM ............................................................
21370 REM                                      TEST DE FIN D'INSTRUCTION
21371 REM ............................................................
21380 IF X=0 THEN T=T+1 : GOTO 21130
21390 FL=0 : RETURN
```

21100 à 21110

Nettoyage de l'écran et affichage du message "ETIQUETTE DE L'INSTRUCTION EN COURS D'EXPLORATION".

21130 à 21135

Nous commençons par le test de fin de programme que nous effectuons sur trois mémoires comme pour le programme TOKEN. De plus, dans le cas d'une fin de programme, nous positionnons le drapeau FL à la valeur −1, et nous sortons du sous-programme.

21150

Les deux premières adresses (T) et (T+1) contiennent la longueur de l'instruction et nous permettent de calculer l'adresse de l'instruction suivante que nous mettons dans la variable AD.

21180

Les deux adresses suivantes (T+2) et (T+3) contiennent l'étiquette de l'instruction qui va être analysée ; nous la calculons, variable ET.

21190

La valeur de l'étiquette en cours d'exploration est affichée en permanence. Cet affichage permet de suivre le déroulement du pro-

gramme et de montrer à l'opérateur que tout se passe normalement car, comme vous pourrez le constater, l'exécution est relativement lente (c'est l'un des principaux inconvénients d'un langage évolué comme le BASIC).

21200 et 21210

Nous incrémentons la variable T une première fois en 21200 de trois unités, puis une seconde fois d'une unité en 21210 pour sauter les quatre premières mémoires que nous venons d'utiliser. L'incrémentation est effectuée en deux fois pour pouvoir réutiliser la seconde dans la boucle d'exploration. Le code ASCII trouvé dans la mémoire d'adresse T est stocké dans la variable X.

21230

Lorsque l'on trouve un REM, soit le token 197, tout ce qui suit dans l'instruction ne peut être qu'une remarque dans le programme ; on ne peut donc y rencontrer de variables. Nous passons alors à l'instruction suivante, ce qui est obtenu en posant T=AD et en effectuant le branchement à l'étiquette 21130.

21250 et 21260

Lorsque, dans une instruction, nous rencontrons le code ASCII 34, soit '', il n'est pas nécessaire d'analyser ce qui suit jusqu'à ce que l'on en rencontre de nouveau un, car ''ce que l'on trouve entre deux '' correspond à du texte.

21250 à 21280

Lorsque nous rencontrons un ordre DATA, nous pouvons aussi sauter ce qui suit jusqu'à ce que l'on rencontre soit le code ASCII zéro (fin d'instruction), soit le code ASCII 1 ('':''). Dans le cas de zéro, nous allons en 21340 continuer et arriver au branchement de fin d'instruction en 21380. Dans le cas de 1, nous continuons l'exploration en effectuant un branchement en 21200 qui incrémente la variable T.

21340

Cette équation logique nous permet de sauter tous les octets contenant la valeur des constantes, valeurs pour lesquelles il pourrait y avoir confusion avec le début d'une variable. Par exemple, pour X

compris entre 13 et 24, la constante occupe un octet ; l'expression logique (X > 13 AND X < 24) vaut alors −1 et −(X > 13 AND X < 24) vaut +1, valeur que nous ajoutons à la variable T. De même pour les autres cas.

21360

Le problème est le même pour la deuxième valeur d'un token double ; donc si nous rencontrons la valeur 255, c'est qu'il s'agit du début d'un token double, nous sautons l'octet suivant.

21380

Dans le cas d'une fin d'instruction, code ASCII zéro, nous incrémentons T et nous allons en 21130 explorer l'instruction suivante.

21390

Si nous arrivons à cet ordre, c'est que le contenu de la mémoire n'est ni un REM, ni de l'alphanumérique, ni une constante, ni un token double ; dans ce cas, nous sortons du sous-programme en positionnant le drapeau FL à 0 pour le signaler.

En conclusion, lorsque ce sous-programme est appelé, on en revient lorsque trois zéros successifs ont été rencontrés, et dans ce cas on positionne la variable FL à −1 pour indiquer que l'exploration est terminée ; ou lorsque la mémoire explorée n'est ni un REM, ni de l'alphanumérique, ni un DATA, ni une fin d'ordre, et dans ce cas la variable FL est positionnée à zéro.

Deuxième sous-programme

Ce sous-programme sert à stocker les variables trouvées et à ne stocker qu'une fois les variables de même nom.

```
21399 REM ..........................................................
21400 REM                                      STOCKAGE DES VARIABLES
21401 REM ..........................................................
21410 I=I+1  :  B$(I)=VA$
21420 FOR J=0  TO  I-1  :  I=I+(VA$=B$(J))  :  NEXT J
21430 VA$=""  :  RETURN
```

21410

I+1 est la valeur du rang de la variable trouvée ; celle-ci est actuellement dans la variable VA$, on la transfère dans B$(I) après avoir fait I=I+1.

gramme et de montrer à l'opérateur que tout se passe normalement car, comme vous pourrez le constater, l'exécution est relativement lente (c'est l'un des principaux inconvénients d'un langage évolué comme le BASIC).

21200 et 21210

Nous incrémentons la variable T une première fois en 21200 de trois unités, puis une seconde fois d'une unité en 21210 pour sauter les quatre premières mémoires que nous venons d'utiliser. L'incrémentation est effectuée en deux fois pour pouvoir réutiliser la seconde dans la boucle d'exploration. Le code ASCII trouvé dans la mémoire d'adresse T est stocké dans la variable X.

21230

Lorsque l'on trouve un REM, soit le token 197, tout ce qui suit dans l'instruction ne peut être qu'une remarque dans le programme ; on ne peut donc y rencontrer de variables. Nous passons alors à l'instruction suivante, ce qui est obtenu en posant T=AD et en effectuant le branchement à l'étiquette 21130.

21250 et 21260

Lorsque, dans une instruction, nous rencontrons le code ASCII 34, soit ", il n'est pas nécessaire d'analyser ce qui suit jusqu'à ce que l'on en rencontre de nouveau un, car "ce que l'on trouve entre deux " correspond à du texte.

21250 à 21280

Lorsque nous rencontrons un ordre DATA, nous pouvons aussi sauter ce qui suit jusqu'à ce que l'on rencontre soit le code ASCII zéro (fin d'instruction), soit le code ASCII 1 (":"). Dans le cas de zéro, nous allons en 21340 continuer et arriver au branchement de fin d'instruction en 21380. Dans le cas de 1, nous continuons l'exploration en effectuant un branchement en 21200 qui incrémente la variable T.

21340

Cette équation logique nous permet de sauter tous les octets contenant la valeur des constantes, valeurs pour lesquelles il pourrait y avoir confusion avec le début d'une variable. Par exemple, pour X

compris entre 13 et 24, la constante occupe un octet ; l'expression logique (X > 13 AND X < 24) vaut alors −1 et −(X > 13 AND X < 24) vaut +1, valeur que nous ajoutons à la variable T. De même pour les autres cas.

21360

Le problème est le même pour la deuxième valeur d'un token double ; donc si nous rencontrons la valeur 255, c'est qu'il s'agit du début d'un token double, nous sautons l'octet suivant.

21380

Dans le cas d'une fin d'instruction, code ASCII zéro, nous incrémentons T et nous allons en 21130 explorer l'instruction suivante.

21390

Si nous arrivons à cet ordre, c'est que le contenu de la mémoire n'est ni un REM, ni de l'alphanumérique, ni une constante, ni un token double ; dans ce cas, nous sortons du sous-programme en positionnant le drapeau FL à 0 pour le signaler.

En conclusion, lorsque ce sous-programme est appelé, on en revient lorsque trois zéros successifs ont été rencontrés, et dans ce cas on positionne la variable FL à −1 pour indiquer que l'exploration est terminée ; ou lorsque la mémoire explorée n'est ni un REM, ni de l'alphanumérique, ni un DATA, ni une fin d'ordre, et dans ce cas la variable FL est positionnée à zéro.

Deuxième sous-programme

Ce sous-programme sert à stocker les variables trouvées et à ne stocker qu'une fois les variables de même nom.

```
21399 REM ..........................................................
21400 REM                                      STOCKAGE DES VARIABLES
21401 REM ..........................................................
21410 I=I+1  :  B$(I)=VA$
21420 FOR J=0  TO  I-1  :  I=I+(VA$=B$(J))  :  NEXT J
21430 VA$=""  :  RETURN
```

21410

I+1 est la valeur du rang de la variable trouvée ; celle-ci est actuellement dans la variable VA$, on la transfère dans B$(I) après avoir fait I=I+1.

21420

On teste si cette variable n'a pas été déjà trouvée en la comparant à tous les B$(J) précédents. Si elle a déjà été trouvée, on diminue alors I d'une unité et elle est oubliée. En effet, l'équation logique (VA$ = B$(J)) vaut −1 si VA$ = B$(J), et zéro dans tous les autres cas, donc I n'est diminué d'une unité que si l'on rencontre un VA$ = B$(J).

21430

Une fois ce travail terminé, on vide VA$ pour le rendre disponible pour la prochaine variable.

Troisième sous-programme

Lorsque l'on a trouvé le début d'une variable, ce sous-programme sert à effectuer la concaténation des lettres ou chiffres qui composent cette variable.

```
21499 REM .........................................................
21500 REM                              CONCATENATION DES VARIABLES
21501 REM .........................................................
21510 VA$=VA$+CHR$(X+128*(X>128))
21520 T=T+1 : X=PEEK(T)
21530 IF PEEK(T-1)<128 THEN 21510
21540 RETURN
```

21510

Effectue la concaténation ; l'équation logique, 128*(X > 128) permet de retirer 128 au code ASCII du dernier caractère de la variable qui, comme nous l'avons vu au Chapitre 3, a été augmentée de 128 par l'interpréteur.

21520

Incrémente T et va chercher le code ASCII suivant.

21530

Vérifie si le code ASCII précédent n'était pas le dernier de la variable (soit X > 128). Si c'est le cas, on sort du sous-programme ; sinon on se branche en 21510 pour continuer la concaténation.

MODULES DE RECHERCHE

Les six modules de recherche sont tous composés de la même manière ; nous expliquerons le premier en détail puis, pour les autres, nous donnerons simplement les différences par rapport à ce module de base.

```
21999 REM ..........................................................
22000 REM                                RECHERCHE DE VARIABLES ENTIERES
22001 REM ..........................................................
22010 NV$="ENTIERES" : VA$="" : GOSUB 21100
22020 IF FL=-1 THEN 28010
22030 T=T-2*(X=3 OR X=13) : IF X<>2 THEN GOSUB 21210 : GOTO 22020
22040 T=T+2 : GOSUB 21510
22050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 22020
22060 GOSUB 21410 : GOSUB 21230 : GOTO 22020
```

22000

REM rappelant le but du module qui suit.

22010

NV$ est la variable dans laquelle nous plaçons le titre de la recherche effectuée pour pouvoir le rappeler au moment de l'édition des résultats. VA$ est la variable dans laquelle nous placerons les caractères composant les variables trouvées ; nous l'initialisons donc à ''rien du tout''. Puis nous appelons le sous-programme situé en 21100, qui est le premier sous-programme, pour calculer l'adresse de l'instruction suivante, le numéro de l'étiquette de l'instruction en cours d'exploration, et pour effectuer une première exploration.

22020

Si nous revenons du sous-programme avec le drapeau FL à -1, c'est que nous sommes arrivés à la fin du programme ; nous allons alors en 28010, vers le menu d'édition.

22030

Dans le cas contraire, FL est égal à zéro, et dans la variable X nous avons le code ASCII de la dernière mémoire explorée. Si ce code ASCII est égal à 3 ou 13, ce sont des variables en virgule flottante ou alphanumériques qui ne nous intéressent pas ; mais nous devons

sauter les deux octets suivants qui correspondent à l'adresse de stockage et qui sont de valeurs quelconques ; ce que nous effectuons par l'équation logique $-2*(X=3 \text{ OR } X=13)$ qui vaut $+1$ pour $X=3$ ou $X=13$.

Nous testons ensuite si ce code ASCII est égal à 2. Si ce n'est pas le cas, il ne s'agit pas du début d'une variable entière ; nous appelons alors le sous-programme d'exploration pour analyser le code ASCII suivant, mais au niveau de l'instruction 21210. En effet, il ne faut pas aller au début de ce sous-programme en 21100, car entre 21100 et 21200 nous traitons les débuts d'instructions. En 21210, nous trouvons l'instruction qui incrémente la variable T d'une unité. Lorsque nous revenons de ce sous-programme, nous retournons en 22020 pour recommencer les mêmes tests.

22040

Lorsque nous arrivons à cette instruction, c'est que le code ASCII contenu dans X est celui d'une variable entière ; nous sautons les deux octets suivants pour les mêmes raisons que précédemment, et nous partons vers le sous-programme de concaténation en 21510.

22050

De retour du sous-programme de concaténation, le résultat se trouve dans VA$, et dans X nous avons la valeur du premier octet situé après la fin de la variable. Dans le cas des variables entières non dimensionnées, il ne doit pas y avoir de "(" après la variable, donc X doit être différent de 40 (code ASCII de "(") ; si X est égal à 40, cette variable ne nous intéresse pas et nous l'effaçons avec VA$="". Ensuite nous appelons de nouveau le programme d'exploration, mais en 21210 car nous avons déjà incrémenté T dans le sous-programme de concaténation et mis le code ASCII de la mémoire dans X. Lorsque nous revenons de ce sous-programme, nous allons en 22020 recommencer les tests.

22060

Lorsque nous arrivons à cette instruction, c'est que nous avons bien trouvé une variable entière non dimensionnée ; nous allons alors dans le sous-programme de stockage en 21410. Le stockage terminé, nous retournons au sous-programme d'exploration en 21230 car T ne doit pas être incrémenté, il l'a déjà été dans le sous-programme de concaténation.

```
22999 REM ...........................................................
23000 REM                              RECHERCHE DE VARIABLES REELLES
23001 REM ...........................................................
23010 NV$="REELLES" : VA$="" : GOSUB 21100
23020 IF FL=-1 THEN 28010
23030 T=T-2*(X=2 OR X=3) : IF X<>13 THEN GOSUB 21210 : GOTO 23020
23040 T=T+2 : GOSUB 21510
23050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 23020
23060 GOSUB 21410 : GOSUB 21230 : GOTO 23020
```

23000 à 23060

Ces instructions sont identiques à celles du module précédent, seul le deuxième chiffre de l'étiquette change ainsi que le rôle joué par les variables entières, en virgule flottante ou alphanumériques, et les codes ASCII associés : 2, 3 et 13.

```
23999 REM ...........................................................
24000 REM                      RECHERCHE DE VARIABLES ALPHANUMERIQUES
24001 REM ...........................................................
24010 NV$="ALPHANUMERIQUES" : VA$="" : GOSUB 21100
24020 IF FL=-1 THEN 28010
24030 T=T-2*(X=2 OR X=13) : IF X<>3 THEN GOSUB 21210 : GOTO 24020
24040 T=T+2 : GOSUB 21510
24050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 24020
24060 GOSUB 21410 : GOSUB 21230 : GOTO 24020
```

24000 à 24060

Ces instructions sont identiques à celles du module précédent ; seul le deuxième chiffre de l'étiquette change, ainsi que le rôle joué par les variables entières, en virgule flottante ou alphanumériques, et les codes ASCII associés : 2, 3 et 13.

```
24999 REM ...........................................................
25000 REM              RECHERCHE DE VARIABLES ENTIERES DIMENSIONNEES
25001 REM ...........................................................
25010 NV$="ENTIERES DIMENSIONNEES" : VA$="" : GOSUB 21100
25020 IF FL=-1 THEN 28010
25030 T=T-2*(X=3 OR X=13) : IF X<>2 THEN GOSUB 21210 : GOTO 25020
25040 T=T+2 : GOSUB 21510
25050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 25020
25060 GOSUB 21410 : GOSUB 21230 : GOTO 25020
```

25000 à 25060

Ces instructions sont identiques à celles du module précédent ; seul le deuxième chiffre de l'étiquette change ainsi que le rôle joué par les variables entières, en virgule flottante ou alphanumériques, et les codes ASCII associés : 2, 3 et 13. Mais maintenant, nous désirons avoir une "(" derrière la variable, puisqu'elle est dimensionnée ; par conséquent, le test fait en 25060 est inversé : c'est X < > 40.

```
25999 REM ..............................................................
26000 REM                  RECHERCHE DE VARIABLES REELLES DIMENSIONNEES
26001 REM ..............................................................
26010 NV$="REELLES DIMENSIONNEES" : VA$="" : GOSUB 21100
26020 IF FL=-1 THEN 28010
26030 T=T-2*(X=2 OR X=3) : IF X<>13 THEN GOSUB 21210 : GOTO 26020
26040 T=T+2 : GOSUB 21510
26050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 26020
26060 GOSUB 21410 : GOSUB 21230 : GOTO 26020
```

26000 à 26060

Ces instructions sont identiques à celles du module précédent ; seul le deuxième chiffre de l'étiquette change ainsi que le rôle joué par les variables entières, en virgule flottante ou alphanumériques, et les codes ASCII associés : 2, 3 et 13. Le test fait, 26060 est toujours X< >40.

```
26999 REM ..............................................................
27000 REM          RECHERCHE DE VARIABLES ALPHANUMERIQUES DIMENSIONNEES
27001 REM ..............................................................
27010 NV$="ALPHANUMERIQUES DIMENSIONNEES" : VA$="" : GOSUB 21100
27020 IF FL=-1 THEN 28010
27030 T=T-2*(X=2 OR X=13) : IF X<>3 THEN GOSUB 21210 : GOTO 27020
27040 T=T+2 : GOSUB 21510
27050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 27020
27060 GOSUB 21410 : GOSUB 21230 : GOTO 27020
```

27000 à 27060

Ces instructions sont identiques à celles du module précédent ; seul le deuxième chiffre de l'étiquette change ainsi que le rôle joué par les variables entières, en virgule flottante ou alphanumériques, et les codes ASCII associés : 2, 3 et 13. Le test fait, 27060 est toujours X< >40.

ÉDITION DES RÉSULTATS

```
27999 REM ..............................................................
28000 REM                                        EDITION DES RESULTATS
28001 REM ..............................................................
28010 CLS
28020 L$="----------------------------------------" : E$="*****"
28030 FOR K=I+1 TO 25 : B$(K)="" : NEXT K
28040 LOCATE 8,4 : PRINT "EDITION DES RESULTATS"
28050 LOCATE 5,10 : PRINT "A L'ECRAN ................. E"
28060 LOCATE 5,14 : PRINT "A L'IMPRIMANTE + ECRAN .... I"
28070 LOCATE 5,20 : PRINT "TAPER VOTRE CHOIX [E/I]"
28080 R$=INKEY$
28090 R$=INKEY$ : IF R$="" THEN 28090 ELSE R=ASC(R$)
28100 IF R=69 THEN 29010
28110 IF R=73 THEN 30010
28120 GOTO 28090
```

```
28999 REM ..........................................................
29000 REM                                                SORTIE ECRAN
29001 REM ..........................................................
29010 CLS
29020 LOCATE 1,1 : PRINT L$ : LOCATE 1,2 : PRINT E$;"      NOMS DES VARIABLES
 ";E$
29030 P=INT((40-LEN(NV$))/2)
29040 PRINT TAB(P);NV$ : PRINT L$
29050 IF I=0 THEN LOCATE 17,12 : PRINT "AUCUNE" : GOTO 29090
29060 FOR J=1 TO I : C$=B$(J)
29070 IF Y=3 OR Y=6 THEN C$=C$+"$"
29080 PRINT C$ : NEXT J
29090 R$=INKEY$ : IF R$="" THEN 29090
29100 GOTO 31010
29999 REM ..........................................................
30000 REM                                   SORTIE ECRAN + IMPRIMANTE
30001 REM ..........................................................
30010 CLS
30020 LOCATE 10,6 : PRINT "ALLUMER L'IMPRIMANTE"
30030 LOCATE 7,14 : PRINT "PUIS TAPEZ SUR UNE TOUCHE"
30040 R$=INKEY$ : IF R$="" THEN 30040
30050 PRINT #8, L$ : PRINT #8, E$;"      NOMS DES VARIABLES    ";E$
30060 P=INT((40-LEN(NV$))/2)
30070 PRINT #8,TAB(P);NV$ : PRINT #8,L$
30080 IF I=0 THEN PRINT #8, : PRINT #8,"AUCUNE" : GOTO 29090
30090 FOR J=1 TO I : C$=B$(J)
30100 IF Y=3 OR Y=6 THEN C$=C$+"$"
30110 PRINT #8,C$ : NEXT J
30120 GOTO 29010
```

Comme pour le programme de recherche de TOKEN, nous donnons la possibilité d'afficher les résultats soit à l'écran, soit sur imprimante plus écran.

28010

Nettoyage de l'écran.

28020

L$ et E$ permettront de simplifier la réalisation du cadre de présentation des résultats.

28030

Remet à zéro les variables B$ qui ont stocké une variable rencontrée deux fois, afin de ne pas la voir s'imprimer deux fois dans la boucle d'écriture (voir boucle d'affichage 29060 à 29080).

28040 à 28070

Présentation des messages de choix du menu.

28080 à 28090

Interrogation du clavier.

28100 et 28120

Aiguillage en fonction du choix de l'opérateur. R=69 code ASCII de E, on effectue un saut à l'ordre 29010 où se trouve le programme d'édition à l'écran. R=73 code ASCII de I, on effectue un saut à l'instruction 30010 où se trouve le programme d'édition sur l'imprimante ; ce programme fera lui-même effectuer un saut en 29010 lorsqu'il aura terminé l'impression, pour faire l'édition à l'écran. Si R est différent de ces deux réponses, retour à l'interrogation du clavier en 28090.

29010

Nettoyage écran.

29020 à 29040

Présentation du titre, NV$ contient le nom du type de variable que l'on vient de rechercher.

29050

Si l'on n'a pas trouvé de variable du type recherché, soit I=0, on affiche ''AUCUNE''.

29060 à 29080

Boucle d'affichage des résultats, deux noms de variables par ligne. Si l'on a trouvé un nombre impair de variables, soit 2*P+1, on affiche aussi avec cette boucle la variable B$(2*P+1) ; c'est pourquoi il est nécessaire de remettre à zéro les B$(J) qui sont au-delà du rang de la dernière variable trouvée.

29090

Boucle d'attente sur le clavier, permettant de regarder les résultats et de passer à la suite en frappant sur n'importe quelle touche.

29100

Saut sur le menu final.

30010

Nettoyage écran.

30020 à 30040

Message pour rappeler qu'il faut mettre l'imprimante sous tension, et boucle d'attente sur le clavier.

30050 à 30070

Impression du titre ; NV$ contient le nom du type de variable que l'on vient de rechercher.

30080

Si l'on n'a pas trouvé de variable du type recherché, soit I=0, on imprime "AUCUNE".

30090 à 30110

Boucle d'impression des résultats, deux noms de variables sont imprimés par ligne.

30120

On se dirige maintenant vers l'édition à l'écran.

Nous terminons par un petit menu de clôture permettant soit de rechercher un autre type de variable, soit de sortir du programme.

```
30999 REM ............................................................
31000 REM                                      QUITTER OU RECOMMENCER
31001 REM ............................................................
31010 CLS
31020 LOCATE 8,10 : PRINT "DESIREZ-VOUS RECHERCHER"
31030 LOCATE 10,14 : PRINT "UNE AUTRE VARIABLE"
31040 LOCATE 15,18 : PRINT "[O/N] :"
31050 R$=INKEY$
31060 R$=INKEY$ : IF R$="" THEN 31060 ELSE R=ASC(R$)
31070 IF R=79 THEN RUN
31080 IF R=78 THEN CLS : END
31090 GOTO 31060
31099 REM ............................................................
```

31010

Nettoyage de l'écran.

31020 à 31040

Affichage des choix du menu et mode d'emploi.

31050 à 31060

Lecture du clavier.

31070

Si l'opérateur désire continuer, réponse O (code ASCII 79) ; on relance le programme par RUN.

31080

Si l'opérateur désire arrêter, réponse N (code ASCII 78) ; on termine avec END.

31090

Retour en 31050 si l'opérateur n'a pas encore répondu, ou s'il a donné une réponse incorrecte.

Vous pouvez bien sûr taper ce programme par morceaux et le sauvegarder au fur et à mesure. Quand vous aurez fini de l'introduire, vous pourrez l'essayer directement ; il s'auto-explorera.

Quand votre programme fonctionnera correctement, vous pourrez passer au chapitre suivant.

LISTING DU PROGRAMME
RECHERCHE DE VARIABLES

```
19994 REM                              PROGRAMME DE RECHERCHE DE VARIABLES
19996 REM                                          TXTTAB=111+  1*256=  367
19998 REM                                           LOMEM=140+ 36*256= 9356
20000 TXTTAB=368
20010 DIM B$(25):FOR I=1 TO 25 : B$(I)="" : NEXT I
20020 CLS
20030 LOCATE 8,1 : PRINT "CHOIX DU TYPE DE VARIABLE"
20040 LOCATE 5,6 : PRINT "ENTIERE ...................... 1"
20050 LOCATE 5,8 : PRINT "REELLE ....................... 2"
20060 LOCATE 5,10 : PRINT "ALPHANUMERIQUE ............... 3"
20070 LOCATE 5,12: PRINT "ENTIERE DIMENSIONNE ......... 4"
20080 LOCATE 5,14 : PRINT "RELLE DIMENSIONNE ........... 5"
20090 LOCATE 5,16 : PRINT "ALPHANUMERIQUE DIMENSIONNE .. 6"
20100 LOCATE 1,20 : PRINT "TAPER LE NUMERO DE VOTRE CHOIX [1/6]"
20110 R$=INKEY$
20120 R$=INKEY$ : IF R$="" THEN 20120 ELSE R=ASC(R$)
20130 IF R<49 OR R>54 THEN 20120 ELSE LOCATE 40,20 : PRINT R$
20140 T=TXTTAB : AD=TXTTAB : I=0
20150 Y=R-48 : ON Y GOTO 22010,23010,24010,25010,26010,27010
20999 REM ...........................................................
21000 REM                                               SOUS PROGRAMMES
21001 REM ...........................................................
21100 CLS : LOCATE 8,10 : PRINT "ETIQUETTE DE L'INSTRUCTION"
21110 LOCATE 10,12 : PRINT "EN COURS D'EXPLORATION"
21119 REM ...........................................................
21120 REM                                     TEST DE FIN DE PROGRAMME
21121 REM ...........................................................
```

```
21130 D=PEEK(T-1) : E=PEEK(T) : F=PEEK(T+1)
21135 IF D=0 AND E=0 AND F=0 THEN FL=-1 : RETURN
21139 REM ..................................................
21140 REM                          ADRESSE DE L'INSTRUCTION SUIVANTE
21141 REM ..................................................
21150 AD=PEEK(T)+256*PEEK(T+1)+AD
21169 REM ..................................................
21170 REM                               ETIQUETTE DE L'INSTRUCTION
21171 REM ..................................................
21180 ET=PEEK(T+2)+256*PEEK(T+3)
21190 LOCATE 18,16 : PRINT ET
21200 T=T+3
21210 T=T+1 : X=PEEK(T)
21219 REM ..................................................
21220 REM                                     ELIMINATION DES REM
21221 REM ..................................................
21230 IF X=197 THEN T=AD : GOTO 21130
21239 REM ..................................................
21240 REM                             ELIMINATION L'ALPHANUMERIQUE
21241 REM ..................................................
21250 IF X=34 THEN 21260 ELSE 21290
21260 T=T+1 : IF PEEK(T)<>34 THEN 21260
21279 REM ..................................................
21280 REM                                    ELIMINATION DES DATA
21281 REM ..................................................
21290 IF X=140 THEN 21300 ELSE 21340
21300 T=T+1 : X=PEEK(T)
```

```
21310 IF X=0 THEN 21340
21320 IF X=1 THEN 21210 ELSE 21300
21329 REM ..............................................................
21330 REM                                    ELIMINATION DES CONSTANTES
21331 REM ..............................................................
21340 T=T-(X>13 AND X<24)-2*(X=25)-3*(X>25 AND X<31)-6*(X=31):X=PEEK(T)
21349 REM ..............................................................
21350 REM                                ELIMINATION DES TOKENS DOUBLES
21351 REM ..............................................................
21360 IF X=255 THEN T=T+2 : X=PEEK(T)
21369 REM ..............................................................
21370 REM                                     TEST DE FIN D'INSTRUCTION
21371 REM ..............................................................
21380 IF X=0 THEN T=T+1 : GOTO 21130
21390 FL=0 : RETURN
21399 REM ..............................................................
21400 REM                                       STOCKAGE DES VARIABLES
21401 REM ..............................................................
21410 I=I+1 : B$(I)=VA$
21420 FOR J=0 TO I-1 : I=I+(VA$=B$(J)) : NEXT J
21430 VA$="" : RETURN
21499 REM ..............................................................
21500 REM                                  CONCATENATION DES VARIABLES
21501 REM ..............................................................
21510 VA$=VA$+CHR$(X+128*(X>128))
21520 T=T+1 : X=PEEK(T)
21530 IF PEEK(T-1)<128 THEN 21510
```

```
21540 RETURN
21999 REM ............................................................
22000 REM                              RECHERCHE DE VARIABLES ENTIERES
22001 REM ............................................................
22010 NV$="ENTIERES" : VA$="" : GOSUB 21100
22020 IF FL=-1 THEN 28010
22030 T=T-2*(X=3 OR X=13) : IF X<>2 THEN GOSUB 21210 : GOTO 22020
22040 T=T+2 : GOSUB 21510
22050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 22020
22060 GOSUB 21410 : GOSUB 21230 : GOTO 22020
22999 REM ............................................................
23000 REM                               RECHERCHE DE VARIABLES REELLES
23001 REM ............................................................
23010 NV$="REELLES" : VA$="" : GOSUB 21100
23020 IF FL=-1 THEN 28010
23030 T=T-2*(X=2 OR X=3) : IF X<>13 THEN GOSUB 21210 : GOTO 23020
23040 T=T+2 : GOSUB 21510
23050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 23020
23060 GOSUB 21410 : GOSUB 21230 : GOTO 23020
23999 REM ............................................................
24000 REM                       RECHERCHE DE VARIABLES ALPHANUMERIQUES
24001 REM ............................................................
24010 NV$="ALPHANUMERIQUES" : VA$="" : GOSUB 21100
24020 IF FL=-1 THEN 28010
24030 T=T-2*(X=2 OR X=13) : IF X<>3 THEN GOSUB 21210 : GOTO 24020
24040 T=T+2 : GOSUB 21510
24050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 24020
```

```
24060 GOSUB 21410 : GOSUB 21230 : GOTO 24020
24999 REM ............................................................
25000 REM                RECHERCHE DE VARIABLES ENTIERES DIMENSIONNEES
25001 REM ............................................................
25010 NV$="ENTIERES DIMENSIONNEES" : VA$="" : GOSUB 21100
25020 IF FL=-1 THEN 28010
25030 T=T-2*(X=3 OR X=13) : IF X<>2 THEN GOSUB 21210 : GOTO 25020
25040 T=T+2 : GOSUB 21510
25050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 25020
25060 GOSUB 21410 : GOSUB 21230 : GOTO 25020
25999 REM ............................................................
26000 REM                 RECHERCHE DE VARIABLES REELLES DIMENSIONNEES
26001 REM ............................................................
26010 NV$="REELLES DIMENSIONNEES" : VA$="" : GOSUB 21100
26020 IF FL=-1 THEN 28010
26030 T=T-2*(X=2 OR X=3) : IF X<>13 THEN GOSUB 21210 : GOTO 26020
26040 T=T+2 : GOSUB 21510
26050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 26020
26060 GOSUB 21410 : GOSUB 21230 : GOTO 26020
26999 REM ............................................................
27000 REM         RECHERCHE DE VARIABLES ALPHANUMERIQUES DIMENSIONNEES
27001 REM ............................................................
27010 NV$="ALPHANUMERIQUES DIMENSIONNEES" : VA$="" : GOSUB 21100
27020 IF FL=-1 THEN 28010
27030 T=T-2*(X=2 OR X=13) : IF X<>3 THEN GOSUB 21210 : GOTO 27020
27040 T=T+2 : GOSUB 21510
27050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 27020
```

```
27060 GOSUB 21410 : GOSUB 21230 : GOTO 27020
27999 REM ...............................................................
28000 REM                                         EDITION DES RESULTATS
28001 REM ...............................................................
28010 CLS
28020 L$="--------------------------------------" : E$="*****"
28030 FOR K=I+1 TO 25 : B$(K)="" : NEXT K
28040 LOCATE 8,4 : PRINT "EDITION DES RESULTATS"
28050 LOCATE 5,10 : PRINT "A L'ECRAN .................. E"
28060 LOCATE 5,14 : PRINT "A L'IMPRIMANTE + ECRAN .... I"
28070 LOCATE 5,20 : PRINT "TAPER VOTRE CHOIX [E/I]"
28080 R$=INKEY$
28090 R$=INKEY$ : IF R$="" THEN 28090 ELSE R=ASC(R$)
28100 IF R=69 THEN 29010
28110 IF R=73 THEN 30010
28120 GOTO 28090
28999 REM ...............................................................
29000 REM                                                  SORTIE ECRAN
29001 REM ...............................................................
29010 CLS
29020 LOCATE 1,1 : PRINT L$ : LOCATE 1,2 : PRINT E$;"      NOMS DES VARIABLES
 ";E$
29030 P=INT((40-LEN(NV$))/2)
29040 PRINT TAB(P);NV$ : PRINT L$
29050 IF I=0 THEN LOCATE 17,12 : PRINT "AUCUNE" : GOTO 29090
29060 FOR J=1 TO I : C$=B$(J)
29070 IF Y=3 OR Y=6 THEN C$=C$+"$"
```

```
29080 PRINT C$ : NEXT J
29090 R$=INKEY$ : IF R$="" THEN 29090
29100 GOTO 31010
29999 REM ..........................................................
30000 REM                                  SORTIE ECRAN + IMPRIMANTE
30001 REM ..........................................................
30010 CLS
30020 LOCATE 10,6 : PRINT "ALLUMER L'IMPRIMANTE"
30030 LOCATE 7,14 : PRINT "PUIS TAPEZ SUR UNE TOUCHE"
30040 R$=INKEY$ : IF R$="" THEN 30040
30050 PRINT #8, L$ : PRINT #8, E$;"     NOMS DES VARIABLES    ";E$
30060 P=INT((40-LEN(NV$))/2)
30070 PRINT #8,TAB(P);NV$ : PRINT #8,L$
30080 IF I=0 THEN PRINT #8, : PRINT #8,"AUCUNE" : GOTO 29090
30090 FOR J=1 TO I : C$=B$(J)
30100 IF Y=3 OR Y=6 THEN C$=C$+"$"
30110 PRINT #8,C$ : NEXT J
30120 GOTO 29010
30999 REM ..........................................................
31000 REM                                     QUITTER OU RECOMMENCER
31001 REM ..........................................................
31010 CLS
31020 LOCATE 8,10 : PRINT "DESIREZ-VOUS RECHERCHER"
31030 LOCATE 10,14 : PRINT "UNE AUTRE VARIABLE"
31040 LOCATE 15,18 : PRINT "[O/N] :"
31050 R$=INKEY$
31060 R$=INKEY$ : IF R$="" THEN 31060 ELSE R=ASC(R$)
```

```
31070 IF R=79 THEN RUN
31080 IF R=78 THEN CLS : END
31090 GOTO 31060
31099 REM ............................................................
```

7

PROGRAMME COMPLET DE DÉVERMINAGE

Aux Chapitres 5 et 6, nous venons de réaliser deux programmes permettant d'effectuer les deux fonctions désirées pour le déverminage d'un programme BASIC. Il ne nous reste plus qu'à réunir ces deux programmes en un seul en y ajoutant un menu permettant d'accéder facilement à l'une ou à l'autre de ces possibilités de recherche :

- soit la recherche d'un ordre BASIC ;
- soit la recherche d'une variable.

Comme vous avez pu le constater, la numérotation des étiquettes des programmes a été effectuée de manière que l'addition (ou plutôt la *réunion*) de ces deux programmes constitue le programme final.

Il restera cependant quelques modifications à apporter concernant le positionnement des ordres DIM en tête de programme de manière à éviter le message d'erreur "REDIMENSIONED ARRAY".

La fonction MERGE — addition de deux programmes — existe sur l'Amstrad, mais elle pose quelques problèmes sur les programmes que nous venons d'écrire (voire sur des programmes beaucoup plus réduits). Nous réaliserons cette fonction très simplement à l'aide de quelques PEEK et POKE, puisque les mouvements de TXTTAB et de LOMEM n'ont plus aucun secret pour vous.

MERGE DE DEUX PROGRAMMES

Nous allons choisir à titre d'exemple deux programmes très courts.

Effectuons un RESET de l'Amstrad avec CTRL+SHIFT+ESC, pour être certains des valeurs présentes en mémoire.

Ensuite, introduisons le programme suivant que nous appelons CH7-P1 :

```
10 REM CH7-P1
20 CLS
30 FOR I=1 TO 2
40 PRINT "CECI EST LE PROGRAMME 1"
50 PRINT "-----------------------"
60 NEXT I
```

Obligatoirement pour ce programme, le TXTTAB est le TXTTAB normal, soit 367.

Le LOMEM vaut 481, ce qui se vérifie en tapant :

```
PRINT PEEK(44675)        Réponse : 225
PRINT PEEK(44676)        Réponse : 1
LOMEM=225+1*256=481
```

Ne faites surtout pas RUN et sauvegardez ce programme en faisant :

```
SAVE "CH7-P1"
```

Effectuons un RESET de l'Amstrad avec CTRL+SHIFT+ESC, pour être certains des valeurs présentes en mémoire.

Introduisons maintenant le Programme 2, que nous appelons CH7-P2.

```
100 REM "CH7-P2"
110 FOR J=1 TO 3
120 PRINT "CECI EST LE PROGRAMME 2"
130 PRINT "++++++++++++++++++++++"
140 NEXT J
150 END
```

Obligatoirement pour ce programme aussi, le TXTTAB est le TXTTAB normal.

Ne faites surtout pas RUN et sauvegardez ce programme en faisant :

```
SAVE "CH7-P2"
```

Effectuons un RESET de l'Amstrad avec CTRL+SHIFT+ESC, pour être certains des valeurs présentes en mémoire.

La fonction MERGE consiste à charger simultanément en mémoire deux programmes (CH7-P1 et CH7-P2) et à les mettre bout à bout pour ne faire qu'un seul programme.

Effectuons maintenant le chargement de CH7-P1 par :

```
LOAD "CH7-P1"
```

Ne faites surtout pas RUN.

Plaçons maintenant le TXTTAB à la valeur du LOMEM actuel moins trois unités, soit 481 – 3 = 478 (ou 222 partie basse et 1 partie haute : 222 + 1 * 256 = 478) :

```
POKE 44673,222
POKE 44674,1
```

Le contenu de la mémoire est actuellement le suivant :

367	0	Début de programme.
368	15	Longueur de la première
369	0	instruction de CH7-P1.
379	10	Étiquette de la première
371	0	instruction de CH7-P1.
372	197	Token de REM.
—		
—		
—		
468	11	Longueur de la dernière
469	0	instruction de CH7-P1.
470	60	Étiquette de la dernière
471	0	instruction de CH7-P1.
472	176	Token de NEXT.
473	32	Espace.
474	13	Variable en virgule flottante.
475	0	Future adresse de stockage
476	0	de la variable.
477	201	Variable I.
478	0	Quatre zéros de fin de programme.
479	0	
480	0	
481	0	

Chargeons alors le programme CH7-P2 par :

```
LOAD "CH7-P2"
```

Le contenu de la mémoire est maintenant le suivant :

367	0	Début de programme.
368	15	Longueur de la première
369	0	instruction de CH7-P1.
379	10	Étiquette de la première
371	0	instruction de CH7-P1.
372	197	Token de REM.

—
—
—

468	11	Longueur de la dernière
469	0	instruction de CH7-P1.
470	60	Étiquette de la dernière
471	0	instruction de CH7-P1.
472	176	Token de NEXT.
473	32	Espace.
474	13	Variable en virgule flottante.
475	0	Future adresse de stockage
476	0	de la variable.
477	201	Variable I.
478	0	Zéro de fin d'instruction.
479	15	Longueur de la première
480	0	instruction de CH7-P2.
481	100	Étiquette de la première
482	0	instruction de CH7-P2.
483	197	Token de REM.

—
—
—

586	6	Longueur de la dernière
587	0	instruction de CH7-P2.
588	150	Étiquette de la dernière
589	0	instruction de CH7-P2.
590	152	Token de END.
591	0	Quatre zéros de fin de programme.
592	0	
593	0	
594	0	

Compte tenu de l'emplacement où nous avons introduit CH7-P2, le raccordement de deux programmes est effectué. Ce que nous pouvons vérifier en tapant maintenant :

```
LIST
```

Et l'on voit apparaître :

```
10 REM CH7-P1
20 CLS
30 FOR I=1 TO 2
40 PRINT "CECI EST LE PROGRAMME 1"
50 PRINT "----------------------"
60 NEXT I
100 REM "CH7-P2"
110 FOR J=1 TO 3
120 PRINT "CECI EST LE PROGRAMME 2"
130 PRINT "++++++++++++++++++++++"
140 NEXT J
150 END
```

On peut enfin faire :

```
RUN
```

Le programme CH7-P1 s'exécute d'abord, puis CH7-P2.

Pour terminer, sauvegardons ce nouveau programme sous le nom CH7-P1 + 2 :

```
SAVE "CH7-P1+2"
```

Cette procédure est très rapide, bien plus rapide que de retaper un programme complet.

Nous vous avons dit de ne pas faire RUN au cours de la manipulation pour deux raisons :

1. pour que les adresses des GOSUB, GOTO et des variables ne soient pas données en valeurs absolues, car l'adressage serait faux après le MERGE ;
2. pour ne pas avoir à modifier tous les LOMEM, ARYTAB... et HIMEM, comme nous avons été obligés de le faire au Chapitre 5.

Nous allons utiliser cette méthode pour la jonction des programmes TOKEN et VARIABLE pour la réalisation d'un programme complet.

RÉALISATION DU PROGRAMME COMPLET DE DÉVERMINAGE

Nous repartons sur un bon pied, c'est-à-dire sur un Amstrad fraîchement initialisé par un RESET. Chargeons le programme TOKEN par :

```
LOAD "TOKEN"
```

puis effectuons la commande suivante :

```
FOR I=367 TO 6334 : PRINT I,PEEK(I) : NEXT I
```

Nous obtenons :

367	0	Début de programme.
368	68	Longueur première instruction
369	0	du programme TOKEN.
370	39	Étiquette première instruction
371	197	du programme TOKEN.
372	32	Espace.
—		
—		
—		
6265	67	Longueur dernière instruction
6266	0	du programme TOKEN.
6267	251	Étiquette dernière instruction
6268	58	du programme TOKEN.
6269	197	Token de REM.
6270	32	Espace.
6271	46	Code ASCII de ".".
—		
—		
—		
6329	46	Code ASCII de ".".
6330	46	Code ASCII de ".".
6331	0	Fin du programme
6332	0	TOKEN.
6333	0	
6334	0	

Modifions maintenant le TXTTAB de la manière précédente en le plaçant au LOMEM−3, soit 6331 (187+24*256=6331) :

```
POKE 44673,187 : POKE 44674,24
```

et chargeons le programme VARIABLE :

```
LOAD "VARIABLE"
```

A la fin du chargement, demandons le LOMEM :

```
PRINT PEEK(44675)+256*PEEK(44676)
```

Réponse : 15320.
Puis faisons :

```
FOR I=367 TO 15320 : PRINT I,PEEK (I) : NEXT I
```

Nous obtenons :

367	0	Début de programme.
368	68	Longueur première instruction
369	0	du programme TOKEN.
370	39	Étiquette première instruction
371	197	du programme TOKEN.
372	32	Espace
—		
—		
—		
6265	67	Longueur dernière instruction
6266	0	du programme TOKEN.
6267	251	Étiquette dernière instruction
6268	58	du programme TOKEN.
6269	197	Token de REM.
6270	32	Espace.
6271	46	Code ASCII de ".".
—		
—		
—		
6329	46	Code ASCII de ".".
6330	46	Code ASCII de ".".
6331	0	Fin du programme TOKEN.
6332	67	Longueur première instruction
6333	0	de VARIABLE.
6334	26	Étiquette première instruction
6335	78	de VARIABLE.
6336	197	Token de REM.
6337	32	Espace.

—
—
—

15251	67	Longueur dernière instruction
15252	0	du programme VARIABLE.
15253	123	Étiquette dernière instruction
15254	121	du programme VARIABLE.
15255	197	Token de REM.
15256	32	Espace.
15257	46	Code ASCII de ".".

—
—
—

15314	46	Code ASCII de ".".
14315	46	Code ASCII de ".".
15317	0	
15318	0	
15319	0	
15320	0	

Les mémoires 6332, 6333 et 6334 sont modifiées, et nous reconnaissons le début du premier ordre du programme VARIABLE. La jonction est effectuée.

Si nous tapons LIST, nous voyons apparaître le programme DEVERMINAGE, qui est la réunion des programmes TOKEN et VARIABLE. Sauvegardons ce programme :

```
SAVE "DEVER"
```

Il ne nous reste plus qu'à compléter ce programme par son menu principal. Il est également nécessaire de replacer les ordres DIM. Nous supprimerons aussi quelques REM qui sont devenues inutiles. C'est ce que nous effectuons ci-après :

```
1000 REM ..........................................................
1010 REM                      PROGRAMME DEVERMINAGE (LONGUEUR  7831)
1020 REM ..........................................................
1100 DIM A(100),A$(10),B$(25)
1110 DATA CLS,DATA,DIM,ELSE,FOR,GOTO,LOCATE,PRINT,READ
```

```
1120 FOR I=1 TO 9 : READ A$(I) : NEXT I
1130 FOR I=1 TO 25 : B$(I)="" : NEXT I
2000 CLS
2010 LOCATE 7,4 : PRINT "CHOIX DE LA RECHERCHE"
2020 LOCATE 6,10 : PRINT "RECHERCHE TOKEN         1"
2030 LOCATE 6,12 : PRINT "RECHERCHE VARIABLE      2"
2040 LOCATE 6,14 : PRINT "FIN DE LA RECHERCHE     3"
2050 LOCATE 4,20 : PRINT "DONNER LA REFERENCE 1,2 OU 3"
2060 R$=INKEY$
2070 R$=INKEY$ : IF R$="" THEN 2070 ELSE 2080
2080 R=ASC(R$) : IF R<49 OR R>51 THEN 2050
2090 IF R=51 THEN END
2100 IF R=49 THEN 10000 ELSE 20000
```

1000 à 1020

REM pour rappeler le titre du programme et son implantation en mémoire.

1100 à 1130

DATA et DIM sont replacés en début de programme, ainsi que l'initialisation des variables.

2000 à 2100

Menu traditionnel en tête de programme. Branchement en 10000 pour la recherche de tokens ; branchement en 20000 pour la recherche de variables, ou fin de l'utilisation du programme.

Nous supprimons la plupart des REM, soit les étiquettes :

```
 9996 -  9998 - 10010 - 10020 - 10030 - 10040 - 10999 - 11000
11001 - 11044 - 11045 - 11046 - 11054 - 11055 - 11056 - 11094
11095 - 11096 - 11104 - 11105 - 11106 - 11124 - 11125 - 11126
11144 - 11145 - 11146 - 11174 - 11175 - 11176 - 11194 - 11195
11196 - 11204 - 11205 - 11206 - 11999 - 12000 - 12001 - 12999
13000 - 13001 - 13999 - 14000 - 14001 - 14999 - 15000 - 15001
15099 - 19996 - 19998 - 20010 - 20999 - 21000 - 21001 - 21119
21120 - 21121 - 21139 - 21140 - 21141 - 21169 - 21170 - 21171
21219 - 21220 - 21221 - 21239 - 21240 - 21241 - 21279 - 21280
21281 - 21329 - 21330 - 21331 - 21349 - 21350 - 21351 - 21369
21370 - 21371 - 21399 - 21400 - 21400 - 21401 - 21499 - 21500
21501 - 21999 - 22000 - 22001 - 22999 - 23000 - 23001 - 23999
24000 - 24001 - 24999 - 25000 - 25001 - 25999 - 26000 - 26001
26999 - 27000 - 27001 - 27999 - 28000 - 28001 - 28999 - 29000
29001 - 29999 - 30000 - 30001 - 30999 - 31000 - 31001 - 31099
```

Attention cependant de ne pas effacer des REM qui serviraient de branchement. L'encombrement du programme est ainsi réduit et la rapidité d'exécution s'en trouve améliorée.

Pour terminer, nous modifions comme suit les instructions : 15070, 15080, 31070 et 31080, pour effectuer un branchement à l'endroit adéquat lorsque l'on désire refaire une exploration de token ou une exploration de variable, ou bien retourner au menu principal.

```
15070 IF R=79 THEN 10000
15080 IF R=78 THEN 2000
31070 IF R=79 THEN 20000
31080 IF R=78 THEN 2000
```

Nous pouvons alors tester ce programme qui s'auto-analyse, après l'avoir sauvegardé bien sûr.

PROCÉDURE DE CHARGEMENT

Nous allons procéder comme nous l'avons expliqué au Chapitre 4.

Tout d'abord, après un RESET de sécurité, nous chargeons normalement le programme à tester. Nous interrogeons le LOMEM ; comme notre programme n'explore pas les valeurs réelles d'un programme, il nous suffit de placer notre programme DEVERMINAGE au-dessus de ce LOMEM, et d'utiliser le HIMEM habituel. Et comme le programme n'a pas été utilisé, il suffit juste de changer la valeur du TXTTAB contenu dans les mémoires d'adresses 44673 et 44674, et la valeur du LOMEM contenu dans les mémoires d'adresses 44675 et 44676 ; le LOMEM est placé trois mémoires au-dessus du TXTTAB choisi.

Si vous voulez explorer un programme qui a déjà été exécuté, vous devez charger le programme à explorer au TXTTAB normal, puis le programme d'exploration à un autre TXTTAB et à un autre HIMEM, repositionner le TXTTAB, le LOMEM et le HIMEM aux valeurs correspondant au premier programme. Vous exécutez le premier programme, puis vous repositionnez aux valeurs du programme d'exploration le TXTTAB, le LOMEM dans ses deux couples de mémoires, ARYTAB, STREND, FRETOP et HIMEM dans ses deux couples de mémoires. Enfin vous pouvez commencer l'exploration.

Cette procédure est plus longue, mais c'est celle que vous devrez utiliser si vous réalisez l'extension d'exploration proposée au Chapitre 8.

8

LES EXTENSIONS DE CES PROGRAMMES

EXTENSION DU PROGRAMME DE RECHERCHE D'ORDRE BASIC

Vous pouvez bien sûr changer les ordres dont on peut effectuer la recherche à l'aide de ce programme ; pour cela, il suffit de changer les DATA, les inscriptions dans le menu et la valeur des tokens correspondants à l'aide de l'Annexe C.

Vous pouvez programmer la recherche des tokens doubles ; pour cela, il sera nécessaire de tester que vous avez trouvé d'abord un octet égal à 255, puis que l'octet suivant a la valeur adéquate.

Vous pouvez aussi compliquer un peu le programme lors de la recherche des GOTO et des GOSUB en stockant, en plus de l'étiquette où apparaît l'ordre, l'étiquette de branchement de l'ordre GOTO ou GOSUB. Il est cependant nécessaire de faire attention au fait que le BASIC de l'Amstrad admet l'ordre IF......THEN 100 ELSE 200 dans lequel le GOTO est implicite.

EXTENSIONS DU PROGRAMME DE RECHERCHE DE VARIABLES

Tout d'abord, il peut être intéressant de mémoriser l'étiquette d'apparition des variables au cours de la recherche. Ou après la recherche, pour une variable donnée, de trouver les numéros d'instructions qui contiennent cette variable.

On peut programmer une recherche des variables et de leurs valeurs associées en calculant lesdites valeurs à partir des codes ASCII trouvés entre LOMEM/ARYTAB, ARYTAB/STREND et HIMEM/FRETOP (voir Chapitre 3).

On peut aussi rechercher non pas les variables qui se trouvent dans les instructions BASIC, mais les variables effectivement utilisées par le programme lorsqu'il est exécuté. Pour cela, il est nécessaire d'explorer, après un RUN, la zone LOMEM/ARYTAB, ARYTAB/STREND et HIMEM/FRETOP, comme nous l'avons fait au Chapitre 3, et de la décoder. Dans ce cas, l'implantation du programme de recherche doit être au-dessus de STREND, et il est nécessaire de protéger la zone HIMEM/FRETOP au moment de l'exécution du programme de recherche, ce qui peut être fait au moment du chargement en repositionnant HIMEM. Nous avons vu au Chapitre 7 comment charger les programmes.

PROGRAMME DE DÉVERMINAGE

```
1000 REM .......................................................
1010 REM                           PROGRAMME DEVERMINAGE (LONGUEUR  7831)
1020 REM .......................................................
1100 DIM A(100),A$(10),B$(25)
1110 DATA CLS,DATA,DIM,ELSE,FOR,GOTO,LOCATE,PRINT,READ
1120 FOR I=1 TO 9 : READ A$(I) : NEXT I
1130 FOR I=1 TO 25 : B$(I)="" : NEXT I
2000 CLS
2010 LOCATE 7,4 : PRINT "CHOIX DE LA RECHERCHE"
2020 LOCATE 6,10 : PRINT "RECHERCHE TOKEN         1"
2030 LOCATE 6,12 : PRINT "RECHERCHE VARIABLE      2"
2040 LOCATE 6,14 : PRINT "FIN DE LA RECHERCHE     3"
2050 LOCATE 4,20 : PRINT "DONNER LA REFERENCE 1,2 OU 3"
2060 R$=INKEY$
2070 R$=INKEY$ : IF R$="" THEN 2070 ELSE 2080
2080 R=ASC(R$) : IF R<49 OR R>51 THEN 2050
2090 IF R=51 THEN END
2100 IF R=49 THEN 10000 ELSE 20000

9994 REM                           PROGRAMME DE RECHERCHE DE TOKEN
10000 TXTTAB=368
10050 FOR I=1 TO 100 : A(I)=0 : NEXT I : CLS
10060 LOCATE 7,1 : PRINT "CHOIX DU TOKEN CHERCHE :"
10070 LOCATE 10,4 : PRINT "CLS             1"
10080 LOCATE 10,6 : PRINT "DATA            2"
10090 LOCATE 10,8 : PRINT "DIM             3"
10100 LOCATE 10,10 : PRINT "ELSE            4"
10110 LOCATE 10,12 : PRINT "FOR             5"
```

```
10120 LOCATE 10,14 : PRINT "GOTO          6"
10130 LOCATE 10,16 : PRINT "LOCATE        7"
10140 LOCATE 10,18 : PRINT "PRINT         8"
10150 LOCATE 10,20 : PRINT "READ          9"
10160 LOCATE 1,23 : PRINT "TAPER LE NUMERO DU TOKEN CHERCHE [1/9]"
10170 R$=INKEY$
10180 R$=INKEY$ : IF R$="" THEN 10180 ELSE R=ASC(R$)
10190 IF R<49 OR R>57 THEN 10180 ELSE LOCATE 40,23 : PRINT R$
10200 TK=0
10210 TK=-138*(R=49)-140*(R=50)-147*(R=51)-151*(R=52)-158*(R=53)
10220 TK=TK-160*(R=54)-169*(R=55)-191*(R=56)-195*(R=57)
10230 RR=R-48
11010 CLS
11020 LOCATE 8,10:PRINT "ETIQUETTE DE L'INSTRUCTION"
11030 LOCATE 10,12:PRINT "EN COURS D'EXPLORATION"
11040 T=TXTTAB : I=0 : AD=TXTTAB
11050 AD=PEEK(T)+256*PEEK(T+1)+AD
11060 ET=PEEK(T+2)+256*PEEK(T+3)
11070 LOCATE 18,16 : PRINT ET
11080 T=T+3
11090 T=T+1 : X=PEEK(T)
11100 IF X=197 THEN T=AD : GOTO 11050
11110 IF X=34 THEN GOTO 11120 ELSE GOTO 11130
11120 T=T+1 : IF PEEK(T)<>34 THEN GOTO 11120
11130 IF X<>13 AND X<>2 AND X<>3 THEN GOTO 11150 ELSE T=T+2
11140 T=T+1 : IF PEEK(T)<129 THEN 11140 ELSE T=T+1 : X=PEEK(T)
11150 IF X=25 THEN T=T+1 : GOTO 11090
11160 IF X>25 AND X<31 THEN T=T+2 : GOTO 11090
```

```
11170 IF X=31 THEN T=T+5 : GOTO 11090
11180 D=PEEK(T): E=PEEK(T+1) : F= PEEK(T+2)
11190 IF D=0 AND E=0 AND F=0 THEN GOTO 12010
11200 IF X=0 THEN T=T+1 : GOTO 11050
11210 IF X=TK THEN I=I+1 : A(I)=ET
11220 GOTO 11090
12010 CLS
12020 LOCATE 8,4 : PRINT "EDITION DES RESULTATS"
12030 LOCATE 5,10 : PRINT "A L'ECRAN ................. E"
12040 LOCATE 5,14 : PRINT "A L'IMPRIMANTE + ECRAN .... I"
12050 LOCATE 5,20 : PRINT "TAPER VOTRE CHOIX [E/I]"
12060 R$=INKEY$
12070 R$=INKEY$ : IF R$="" THEN 12070 ELSE R=ASC(R$)
12080 IF R=69 THEN 13010
12090 IF R=73 THEN 14010
12100 GOTO 12070
13010 CLS
13020 LOCATE 3,1 : PRINT "ETIQUETTE DES INSTRUCTIONS CONTENANT"
13030 LOCATE 17,3 : PRINT A$(RR)
13040 IF I=0 THEN LOCATE 17,10 : PRINT "AUCUN" : GOTO 13080
13050 FOR J=1 TO I STEP 4
13060 PRINT USING "#######";A(J);A(J+1);A(J+2);A(J+3)
13070 NEXT J
13080 R$=INKEY$
13090 R$=INKEY$ : IF R$="" THEN 13090
13100 GOTO 15010
14010 CLS
14020 LOCATE 8,8 : PRINT "ALLUMER L'IMPRIMANTE"
```

```
14030 LOCATE 7,12 : PRINT "PUIS ENFONCER <RETURN>"
14040 LOCATE 17,16 : INPUT Z$
14050 PRINT #8,"ETIQUETTE DES INSTRUCTIONS CONTENANT"
14060 PRINT #8,:PRINT #8,A$(RR):PRINT #8,:PRINT #8,
14070 IF I=0 THEN PRINT #8,"      --- AUCUN ---" : GOTO 13010
14080 FOR J=1 TO I STEP 4
14090 PRINT #8,A(J),A(J+1),A(J+2),A(J+3)
14100 NEXT J
14110 GOTO 13010
15010 CLS
15020 LOCATE 8,6 : PRINT "DESIREZ-VOUS EXPLORER"
15030 LOCATE 11,10 : PRINT "UN AUTRE TOKEN"
15040 LOCATE 12,15 : PRINT "[OUI/NON] :"
15050 R$=INKEY$
15060 R$=INKEY$ : IF R$="" THEN 15060 ELSE R=ASC(R$)
15070 IF R=79 THEN 10000
15080 IF R=78 THEN 2000
15090 GOTO 15040
19994 REM                         PROGRAMME DE RECHERCHE DE VARIABLES
20000 TXTTAB=368
20020 CLS
20030 LOCATE 8,1 : PRINT "CHOIX DU TYPE DE VARIABLE"
20040 LOCATE 5,6 : PRINT "ENTIERE ...................... 1"
20050 LOCATE 5,8 : PRINT "REELLE ....................... 2"
20060 LOCATE 5,10 : PRINT "ALPHANUMERIQUE ............... 3"
20070 LOCATE 5,12: PRINT "ENTIERE DIMENSIONNE ......... 4"
20080 LOCATE 5,14 : PRINT "RELLE DIMENSIONNE ........... 5"
20090 LOCATE 5,16 : PRINT "ALPHANUMERIQUE DIMENSIONNE .. 6"
```

```
20100 LOCATE 1,20 : PRINT "TAPER LE NUMERO DE VOTRE CHOIX [1/6]"
20110 R$=INKEY$
20120 R$=INKEY$ : IF R$="" THEN 20120 ELSE R=ASC(R$)
20130 IF R<49 OR R>54 THEN 20120 ELSE LOCATE 40,20 : PRINT R$
20140 T=TXTTAB : AD=TXTTAB : I=0
20150 Y=R-48 : ON Y GOTO 22010,23010,24010,25010,26010,27010
21100 CLS : LOCATE 8,10 : PRINT "ETIQUETTE DE L'INSTRUCTION"
21110 LOCATE 10,12 : PRINT "EN COURS D'EXPLORATION"
21130 D=PEEK(T-1) : E=PEEK(T) : F=PEEK(T+1)
21135 IF D=0 AND E=0 AND F=0 THEN FL=-1 : RETURN
21150 AD=PEEK(T)+256*PEEK(T+1)+AD
21180 ET=PEEK(T+2)+256*PEEK(T+3)
21190 LOCATE 18,16 : PRINT ET
21200 T=T+3
21210 T=T+1 : X=PEEK(T)
21230 IF X=197 THEN T=AD : GOTO 21130
21250 IF X=34 THEN 21260 ELSE 21290
21260 T=T+1 : IF PEEK(T)<>34 THEN 21260
21290 IF X=140 THEN 21300 ELSE 21340
21300 T=T+1 : X=PEEK(T)
21310 IF X=0 THEN 21340
21320 IF X=1 THEN 21210 ELSE 21300
21340 T=T-(X>13 AND X<24)-2*(X=25)-3*(X>25 AND X<31)-6*(X=31):X=PEEK(T)
21360 IF X=255 THEN T=T+2 : X=PEEK(T)
21380 IF X=0 THEN T=T+1 : GOTO 21130
21390 FL=0 : RETURN
21410 I=I+1 : B$(I)=VA$
21420 FOR J=0 TO I-1 : I=I+(VA$=B$(J)) : NEXT J
```

```
21430 VA$="" : RETURN
21510 VA$=VA$+CHR$(X+128*(X>128))
21520 T=T+1 : X=PEEK(T)
21530 IF PEEK(T-1)<128 THEN 21510
21540 RETURN
22010 NV$="ENTIERES" : VA$="" : GOSUB 21100
22020 IF FL=-1 THEN 28010
22030 T=T-2*(X=3 OR X=13) : IF X<>2 THEN GOSUB 21210 : GOTO 22020
22040 T=T+2 : GOSUB 21510
22050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 22020
22060 GOSUB 21410 : GOSUB 21230 : GOTO 22020
23010 NV$="REELLES" : VA$="" : GOSUB 21100
23020 IF FL=-1 THEN 28010
23030 T=T-2*(X=2 OR X=3) : IF X<>13 THEN GOSUB 21210 : GOTO 23020
23040 T=T+2 : GOSUB 21510
23050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 23020
23060 GOSUB 21410 : GOSUB 21230 : GOTO 23020
24010 NV$="ALPHANUMERIQUES" : VA$="" : GOSUB 21100
24020 IF FL=-1 THEN 28010
24030 T=T-2*(X=2 OR X=13) : IF X<>3 THEN GOSUB 21210 : GOTO 24020
24040 T=T+2 : GOSUB 21510
24050 IF X=40 THEN VA$="" : GOSUB 21230 : GOTO 24020
24060 GOSUB 21410 : GOSUB 21230 : GOTO 24020
25010 NV$="ENTIERES DIMENSIONNEES" : VA$="" : GOSUB 21100
25020 IF FL=-1 THEN 28010
25030 T=T-2*(X=3 OR X=13) : IF X<>2 THEN GOSUB 21210 : GOTO 25020
25040 T=T+2 : GOSUB 21510
25050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 25020
```

```
25060 GOSUB 21410 : GOSUB 21230 : GOTO 25020
26010 NV$="REELLES DIMENSIONNEES" : VA$="" : GOSUB 21100
26020 IF FL=-1 THEN 28010
26030 T=T-2*(X=2 OR X=3) : IF X<>13 THEN GOSUB 21210 : GOTO 26020
26040 T=T+2 : GOSUB 21510
26050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 26020
26060 GOSUB 21410 : GOSUB 21230 : GOTO 26020
27010 NV$="ALPHANUMERIQUES DIMENSIONNEES" : VA$="" : GOSUB 21100
27020 IF FL=-1 THEN 28010
27030 T=T-2*(X=2 OR X=13) : IF X<>3 THEN GOSUB 21210 : GOTO 27020
27040 T=T+2 : GOSUB 21510
27050 IF X<>40 THEN VA$="" : GOSUB 21230 : GOTO 27020
27060 GOSUB 21410 : GOSUB 21230 : GOTO 27020
28010 CLS
28020 L$="----------------------------------------" : E$="*****"
28030 FOR K=I+1 TO 25 : B$(K)="" : NEXT K
28040 LOCATE 8,4 : PRINT "EDITION DES RESULTATS"
28050 LOCATE 5,10 : PRINT "A L'ECRAN ................. E"
28060 LOCATE 5,14 : PRINT "A L'IMPRIMANTE + ECRAN .... I"
28070 LOCATE 5,20 : PRINT "TAPER VOTRE CHOIX [E/I]"
28080 R$=INKEY$
28090 R$=INKEY$ : IF R$="" THEN 28090 ELSE R=ASC(R$)
28100 IF R=69 THEN 29010
28110 IF R=73 THEN 30010
28120 GOTO 28090
29010 CLS
29020 LOCATE 1,1 : PRINT L$ : LOCATE 1,2 : PRINT E$;"    NOMS DES VARIABLE
 ";E$
```

```
29030 P=INT((40-LEN(NV$))/2)
29040 PRINT TAB(P);NV$ : PRINT L$
29050 IF I=0 THEN LOCATE 17,12 : PRINT "AUCUNE" : GOTO 29090
29060 FOR J=1 TO I : C$=B$(J)
29070 IF Y=3 OR Y=6 THEN C$=C$+"$"
29080 PRINT C$ : NEXT J
```

A

CODES ASCII PRODUITS PAR LE CLAVIER

Valeur décimale	Touche
0	CTRL+@
1	CTRL+A
2	CTRL+B
3	CTRL+C
4	CTRL+D
5	CTRL+E
6	CTRL+F
7	CTRL+G
8	CTRL+H
9	TAB ou CTRL+I
10	CTRL+J
11	CTRL+K
12	CTRL+L
13	ENTER ou CTRL+M
14	CTRL+N
15	CTRL+O
16	CLR ou CTRL+P
17	CTRL+Q
18	CTRL+R
19	CTRL+S
20	CTRL+T
21	CTRL+U
22	CTRL+V
23	CTRL+W
24	CTRL+X
25	CTRL+Y
26	CTRL+Z
27	CTRL+[
28	CTRL+\
29	CTRL+]

Valeur décimale	Touche
30	CTRL+↑
31	CTRL+0
32	ESPACE
33	!
34	''
35	#
36	$
37	%
38	&
39	´
40	(
41	)
42	
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;

Valeur décimale	Touche
60	<
61	=
62	>
63	?
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	↑
95	—
96	`
97	a
98	b
99	c

Valeur décimale	Touche
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	
124	|
125	
126	CTRL + 2
127	DELETE
128	
129	
130	
131	
132	
133	
134	
135	
136	
137	
138	
139	

Valeur déci-male	Touche	Valeur déci-male	Touche
140		180	
141		181	
142		182	
143		183	
144		184	
145		185	
146		186	
147		187	
148		188	
149		189	
150		190	
151		191	
152		192	
153		193	
154		194	
155		195	
156		196	
157		197	
158		198	
159		199	
160		200	
161		201	
162		202	
163	#	203	
164		204	
165		205	
166		206	
167		207	
168		208	
169		209	
170		210	
171		211	
172		212	
173		213	
174		214	
175		215	
176		216	
177		217	
178		218	
179		219	

Valeur décimale	Touche	Valeur décimale	Touche
220		238	
221		239	
222		240	↑
223		241	↓
224	COPY	242	←
225	CTRL+TAB	243	→
226		244	
227		245	
228		246	
229		247	
230		248	CTRL+ ↑
231		249	CTRL+ ↓
232		250	CTRL+ ←
233		251	CTRL+ →
234		252	
235		253	
236		254	
237		255	

Remarque

Certains codes ASCII ne correspondent à aucune touche ou combinaison de touches, ils peuvent avoir des fonctions, par exemple pour envoyer des caractères ''contrôle'' à l'imprimante, et ils ne peuvent être créés qu'à l'aide de la fonction CHR$. L'interpréteur ou l'ordre POKE peuvent bien sûr créer n'importe quel code ASCII dans les mémoires.

B

TOKENS DE L'AMSTRAD DANS L'ORDRE NUMÉRIQUE

128	AFTER
129	AUTO
130	BORDER
131	CALL
132	CAT
133	CHAIN
134	CLEAR
135	CLG
136	CLOSE IN
137	CLOSE OUT
138	CLS
139	CONT
140	DATA
141	DEF
142	DEFINT
143	DEFREAL
144	DEFSTR
145	DEG
146	DELETE
147	DIM
148	DRAW
149	DRAWR
150	EDIT
151	ELSE
152	END
153	ENT
154	ENV
155	ERASE
156	ERROR
157	EVERY
158	FOR
159	GOSUB
160	GOTO
161	IF
162	INK
163	INPUT
164	KEY
165	LET
166	LINE
167	LIST
168	LOAD
169	LOCATE
170	MEMORY
171	MERGE
172	MID$
173	MODE
174	MOVE
175	MOVER
176	NEXT
177	NEW
178	ON
179	ON BREAK
180	ON ERROR GOTO
181	ON SQ
182	OPENIN
183	OPENOUT
184	ORIGIN
185	OUT
186	PAPER
187	PEN
188	PLOT
189	PLOTR
190	POKE
191	PRINT
192	'
193	RAD

194	RANDOMIZE	238	>
195	READ	239	=
196	RELEASE	240	> =
197	REM	241	<
198	RENUM	242	< >
199	RESTORE	243	< =
200	RESUME	244	+
201	RETURN	245	−
202	RUN	246	*
203	SAVE	247	/
204	SOUND	248	^
205	SPEED	249	
206	STOP	250	AND
207	SYMBOL	251	MOD
208	TAG	252	OR
209	TAGOFF	253	XOR
210	TROFF	254	NOT
211	TRON	255	
212	WAIT	255-0	ABS
213	WEND	255-1	ASC
214	WHILE	255-2	ATN
215	WIDHT	255-3	CHR$
216	WINDOW	255-4	CINT
217	WRITE	255-5	COS
218	ZONE	255-6	CREAL
219	DI	255-7	EXP
220	EI	255-8	FIX
221		255-9	FRE
222		255-10	INKEY
223		255-11	INP
224		255-12	INT
225		255-13	
226		255-14	LEN
227	ERL	255-15	LOG
228	FN	255-16	LOG10
229	SPC	255-17	LOWER$
230	STEP	255-18	PEEK
231	SWAP	255-19	REMAIN
232		255-20	SGN
233		255-21	SIN
234	TAB	255-22	SPACE$
235	THEN	255-23	SQ
236	TO	255-24	SQR
237	USING	255-25	STR$

255-26	TAN	255-70	TIME
255-27	UNT	255-71	XPOS
255-28	UPPER$	255-72	YPOS
255-29	VAL	255-73	
255-30		255-74	
255-31		255-75	
255-32		255-76	
255-33		255-77	
255-34		255-78	
255-35		255-79	
255-36		255-80	
255-37		255-81	
255-38		255-82	
255-39		255-83	
255-40		255-84	
255-41		255-85	
255-42		255-86	
255-43		255-87	
255-44		255-88	
255-45		255-89	
255-46		255-90	
255-47		255-91	
255-48		255-92	
255-49		255-93	
255-50		255-94	
255-51		255-95	
255-52		255-96	
255-53		255-97	
255-54		255-98	
255-55		255-99	
255-56		255-100	
255-57		255-101	
255-58		255-102	
255-59		255-103	
255-60		255-104	
255-61		255-105	
255-62		255-106	
255-63		255-107	
255-64	EOF	255-108	
255-65	ERR	255-109	
255-66	HIMEM	255-110	
255-67	INKEY$	255-111	
255-68	PI	255-112	
255-69	RND	255-113	BIN$

255-114	
255-115	HEX$
255-116	INSTR
255-117	LEFT$
255-118	MAX
255-119	MIN
255-120	POS
255-121	RIGHT$
255-122	ROUND
255-123	STRING$
255-124	TEST
255-125	TESTR
255-127	
255-127	VPOS

C

TOKENS DE L'AMSTRAD DANS L'ORDRE ALPHABÉTIQUE

ABS	255-0
AFTER	128
AND	250
ASC	255-1
AUTO	129
BIN$	255-113
BORDER	130
CALL	131
CAT	132
CHAIN	133
CHR$	255-3
CINT	255-4
CLEAR	134
CLG	135
CLOSEIN	136
CLOSEOUT	137
CLS	138
CONT	139
COS	255-5
CREAL	255-6
DATA	140
DEF	141
DEFINT	142
DEFREAL	143
DEFSTR	144
DEG	145
DELETE	146
DI	119
DIM	147
DRAW	148
DRAWR	149
EDIT	150
EI	220
ELSE	151
END	152
ENT	153
ENV	154
EOF	255-64
ERASE	155
ERL	227
ERR	255-65
ERROR	156
EVERY	157
EXP	255-7
FIX	255-8
FN	228
FOR	158
FRE	255-9
GOSUB	159
GOTO	160
HEX$	255-115
HIMEM	255-66
IF	161
INK	162
INKEY$	255-67
INP	255-11
INPUT	163
INTR	255-116
INT	255-12
JOY	255-13
KEY	164
LEFT$	255-117
LEN	255-14
LET	165
LINE	166
LIST	167

LOAD	168	RIGHT$	255-121
LOCATE	169	RND	255-69
LOG	255-15	ROUND	255-122
LOG1O	255-16	RUN	202
LOWER$	255-17	SAVE	203
MAX	255-118	SGN	255-20
MEMORY	170	SIN	255-21
MERGE	171	SOUND	204
MID$	172	SPACE$	255-22
MIN	255-119	SPC	229
MOD	251	SPEED	205
MODE	173	SQ	255-23
MOVE	174	SQR	255-24
MOVER	175	STEP	230
NEXT	176	STOP	206
NEW	177	STR$	255-25
NOT	254	STRING$	255-123
ON	178	SWAP	231
ON BREAK	179	SYMBOL	207
ON ERROR GOTO	180	TAB	234
ON SQ	181	TAG	208
OPENIN	182	TAGOFF	209
OPENOUT	183	TAN	255-26
OR	252	TEST	255-124
ORIGIN	184	TESTR	255-125
OUT	185	THEN	235
PAPER	186	TIME	255-70
PEEK	255-18	TO	236
PEN	187	TROFF	210
PI	255-68	TRON	211
PLOT	188	UNT	255-27
PLOTR	189	UPPER$	255-28
POKE	190	USING	237
POS	255-12	VAL	255-29
PRINT	191	VPOS	255-127
RAD	193	WAIT	212
RANDOMIZE	194	WEND	213
READ	195	WHILE	214
RELEASE	196	WIDHT	215
REMAIN	255-19	WINDOW	216
RENUM	198	WRITE	217
RESTORE	199	XOR	253
RESUME	200	XPOS	255-71
RETURN	201	YPOS	255-72

ZONE	218
>	238
>=	240
<	241
< >	242
<=	243
+	244
−	245
*	246
/	247
^	248
'	192

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par* RODNAY ZAKS,
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par* RODNAY ZAKS,
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par* RODNAY ZAKS,
636 pages, Réf. 0040

TECHNIQUES D'INTERFACE aux microprocesseurs *par* AUSTIN LESEA ET RODNAY ZAKS,
450 pages, Réf. 0039

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par* JOËL PONCET,
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par* PIERRE LE BEUX,
140 pages, Réf. 369

LA SOLUTION RS-232 *par* JOE CAMPBELL,
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par* PIERRICK BOURGAULT,
198 pages, Réf. 0119

ROBOTS - CONSTRUCTION, PROGRAMMATION *par* FERNAND ESTEVES,
400 pages, Réf. 0130

ALGORITHMES *par* PIERRE BEAUFILS ET WOLFRAM LUTHER,
296 pages, Réf. 0149

BASIC

VOTRE PREMIER PROGRAMME BASIC *par* RODNAY ZAKS,
208 pages, Réf. 263

INTRODUCTION AU BASIC *par* PIERRE LE BEUX,
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par* JEAN-PIERRE LAMOITIER,
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par* XUAN TUNG BUI,
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par* ALAN R. MILLER,
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par* RICHARD MATEOSIAN,
352 pages, Réf. 0129

JEUX D'ORDINATEUR EN BASIC *par* DAVID H. AHL,
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par* DAVID H. AHL,
204 pages, Réf. 247

FICHIERS EN BASIC *par* ALAN SIMPSON,
256 pages, Réf. 0102

TECHNIQUES DE PROGRAMMATION EN BASIC *par* S. CROSMARIE, M. PERRON ET D. PHILIPPINE
152 pages, Réf. 0124

PASCAL

INTRODUCTION AU PASCAL *par* PIERRE LE BEUX,
496 pages, Réf. 0030

LE PASCAL PAR LA PRATIQUE *par* PIERRE LE BEUX ET HENRI TAVERNIER,
562 pages, Réf. 361

LE GUIDE DU PASCAL *par* JACQUES TIBERGHIEN,
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par* ALAN R. MILLER,
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par* PIERRE LE BEUX,
366 pages, Réf. 360

INTRODUCTION A C *par* BRUCE HUNTER,
312 pages, Réf. 0092

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par* PIERRE MONSAUT,
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par* RODNAY ZAKS,
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par* STANLEY R. TROST,
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par* NORBERT RIMOUX,
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par* GEORGES FAGOT-BARRALY,
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par* RODNAY ZAKS,
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par* STANLEY R. TROST,
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par* PIERRE MONSAUT,
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

AMSTRAD, GUIDE DU GRAPHISME *par James Wynford,*
208 pages, Réf. 0141

AMSTRAD CP/M 2.2 *par Anatole D'Hardencourt,*
248 pages, Réf. 0156

AMSTRAD ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
160 pages, Réf. 0167

AMSTRAD MULTIPLAN *de Microsoft,*
496 pages, Réf. 1111

AMSTRAD, CRÉER DE NOUVELLES INSTRUCTIONS
par Jean-Claude Despoine,
144 pages, Réf. 0176

AMSTRAD ASTROCALC
par Gérard Blanc et Philippe Destrebecq,
168 pages, Réf. 0162

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II,
Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert,*
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

INTRODUCTION A MAC PASCAL *par Pierre Le Beux,*
416 pages, Réf. 0145

MACINTOSH POUR LA PRESSE, L'EDITION ET
LA PUBLICITE *par Bernard Le Du,*
160 pages, Réf. 0173

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn,*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar,*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis,*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Lippman,
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford,*
320 pages, Réf. 357

GUIDE DE PC DOS *par Richard A. King,*
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre,*
272 pages, Réf. 0118

MO 5, STATIQUE, DYNAMIQUE, ELECTRONIQUE, PROGRAMMES DE PHYSIQUE EN BASIC
par Beaufils, Lamarche et Muggianu,
240 pages, Réf. 0148

MO 5, PROGRAMMES D'ELECTRONIQUE EN BASIC
par Beaufils, Delusurieux, Do, Romanacce,
312 pages, Réf. 0143

MO 5, OPTIQUE, THERMODYNAMIQUE, CHIMIE
par P. Beaufils, M. Lamarche, Y. Muggianu,
224 pages, Réf. 0161

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

MSX, PROGRAMMES EN LANGAGE MACHINE
par Steeve Webb,
112 pages, Réf. 0153

MSX, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
216 pages, Réf. 0144

MSX, GUIDE DU BASIC *par Michel Laurent,*
264 pages, Réf. 0155

MSX, JEUX EN ASSEMBLEUR *par Eric Ravis*
112 pages, Réf. 0170

MSX, ROUTINES GRAPHIQUES EN ASSEMBLEUR
par Steeve Webb
88 pages, Réf. 0154

MSX, TECHNIQUES DE PROGRAMMATION DES JEUX EN ASSEMBLEUR
par Georges Fagot-Barraly,
176 pages, Réf. 0178

MSX ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
157 pages, Réf. 0168

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir,
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

TO 7 / MO 5, GUIDE DU BASIC
par Jean-Louis Greco et Michel Laurent,
288 pages, Réf. 0158

TO 7 / MO 5, GUIDE DU GRAPHISME
par Michel Lamarche et Yves Muggianu,
240 pages, Réf. 0172

TO 7 / MO 5 ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
160 pages, Réf. 0169

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost,
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 0058

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 0181

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088 *par François Retoreau,*
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

INTRODUCTION A UNIX *par John D. Halamka,*
240 pages, Réf. 0098

GUIDE DE PRODOS
par Pierre Beaufils et Wolfram Luther,
248 pages, Réf. 0146

APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 61 80 2-0
Telex : 08588163



Paris • Berkeley • Düsseldorf

Achevé d'imprimer le 10 mars 1986 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 1er trimestre 1986 - Imprimeur n° 1960

Dès qu'un programme BASIC atteint une certaine taille, il se pose le problème de la mise au point et de la recherche des erreurs : le "déverminage".
Certaines sortes de "déverminage" peuvent être faites à l'aide de logiciels, par exemple liste des variables utilisées, numéro des étiquettes où elles apparaissent ; liste des GOTO ou des GOSUB ; recherche de la valeur d'une variable...
Ce livre étudie tout d'abord la manière dont sont stockés, dans la mémoire d'un micro-ordinateur, un programme BASIC et ses variables associées, puis il vous guide pas à pas dans la réalisation d'un logiciel de déverminage. Il se termine par le listing d'un tel logiciel comportant :

- Recherche des étiquettes où apparaissent des ordres BASIC choisis
- Recherche des variables utilisées dans les instructions.



SYBEX

AMSTRAD MISE AU POINT DES PROGRAMMES BASIC



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/